An Improved Technique for Modeling and Control of Flexible Structures

A Thesis
Presented to
The Academic Faculty

by

Ryan W. Krauss

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

George W. Woodruff School of Mechanical Engineering
Georgia Institute of Technology
August 2006

An Improved Technique for Modeling and Control of Flexible Structures

Approved by:

Dr. Wayne J. Book, Advisor
School of Mechanical Engineering
Georgia Institute of Technology

Dr. James Craig
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Al Ferri
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Dewey Hodges
School of Aerospace Engineering
Georgia Institute of Technology

Dr. William Singhose
School of Mechanical Engineering
Georgia Institute of Technology

Date Approved: June 23, 2006

To Missy,

without your support I could

not have pursued this dream. Thank you.

ACKNOWLEDGEMENTS

I would first of all like to thank my wife Missy for her love and support during this time. I could not have pursued this dream without her help.

I would also like to thank Dr. Wayne Book for his his wisdom and character in guiding this work, as well as for the freedom and financial support to pursue whatever interested me in this area. He has been a mentor to me and it has been a pleasure working for him.

My thanks also go to Drs. Al Ferri, Bill Singhose, James Craig, and Dewey Hodges for sharing their wisdom and their time in guiding this work and serving on the thesis committee.

I want to thank JD Huggins for his help with the experimental side of this work and being a great resource for all hydraulic and electrical problems.

I would like to thank all of the students in the IMDL for letting me bounce ideas off of them and for their friendship and camaraderie. I would like to specifically thank Dr. Davin Swanson, Dr. LJ Tognetti, Joe Frankel, Matt Kontz, Amir Shenouda, Ben Black, and Dr. Haihong Zhu.

Finally, thanks to my parents and all of our family and friends who believed in me and loved and supported us through this time.

TABLE OF CONTENTS

DEDICATION . . . . . **iii**

ACKNOWLEDGEMENTS . . . . . **iv**

LIST OF TABLES . . . . . **x**

LIST OF FIGURES . . . . . **xi**

SUMMARY . . . . . **xix**

I INTRODUCTION **1**

1.1 Background/Problem Statement 1

1.1.1 Problem Statement 2

1.1.2 An Assertion 3

1.1.3 Claim 4

1.1.4 Contributions 4

1.2 Literature Review 6

1.2.1 Modeling of Flexible Robots 6

1.2.2 Controls 10

1.2.3 Progression of the Work at Georgia Tech 13

1.3 Introduction to the Transfer Matrix Method (TMM) 15

1.3.1 TMM Mode Shape Determination 17

1.3.2 Bode Analysis 18

1.3.3 System Model 19

1.3.4 Classical Analysis 21

1.3.5 More Complicated Systems 22

II HYDRAULIC ACTUATOR MODELING **23**

2.1 Introduction 23

2.2 Initial Model 23

2.3 An Actuator Model with Intrinsic Velocity Feedback 26

2.4 Torque Experiments 28

2.5 Angular Velocity Source in Series with a Torsional Spring/Damper 33

III NON-COLLOCATED FEEDBACK MODELING 36

3.1 Introduction 36

3.2 Position Control - without Vibration Suppression 37

3.3 With Vibration Suppression 40

3.4 Experimental Verification 45

3.4.1 System ID 45

3.4.2 θ Feedback Modeling 49

3.4.3 Acceleration Feedback 51

3.4.4 Effectiveness of Vibration Suppression 53

IV MODELING ARBITRARY THREE DIMENSIONAL POSES OF FLEXIBLE ROBOTS 55

4.1 Introduction 55

4.2 Beam Derivations 55

4.2.1 Bending 55

4.2.2 Axial and Torsional Vibration 60

4.2.3 Mixed States 62

4.2.4 Final Beam Transfer Matrix 64

4.3 Rigid Body Matrix 65

4.3.1 Final Rigid Body Matrix/Mixed States 67

4.4 Rotation Matrix 68

4.5 Numerical Verification 69

4.5.1 TMM Analysis Procedure 69

4.5.2 Analysis Details 71

4.5.3 Natural Frequencies 72

4.5.4 Mode Shapes 72

V CONTROL DESIGN USING THE TMM 79

5.1 Introduction 79

5.2 Automated Bode Design and Optimization 79

5.2.1 θ Feedback Design 80

5.2.2 Acceleration Feedback Design 84

5.2.3 Compensator Design 87

5.2.4 Simultaneous Design . . . 88
5.2.5 Time-Domain Response . . . 91
5.3 Direct Control Design . . . 94
5.3.1 Computer Algorithm and Usage . . . 95
5.3.2 Analysis Example . . . 96
5.3.3 Verification . . . 98
5.3.4 Symbolic Control Design . . . 99
5.3.5 Comparison to the Method of Book and Majette . . . 102
5.3.6 Pole-Placement Design for SAMII . . . 103
5.3.7 SAMII Optimization . . . 111
5.4 Generalizing the Control Design Strategy . . . 114
5.4.1 Avoiding Saturation . . . 115
5.4.2 Pole-Placement Strategy . . . 118
5.5 Dependence on fmin . . . 119
5.6 Robustness . . . 120
VI SOFTWARE DESIGN . . . 131
6.1 Introduction . . . 131
6.1.1 Overview . . . 131
6.1.2 Software Usage . . . 132
6.1.3 Novel Features . . . 132
6.2 The Benefits of an Object-Oriented Approach . . . 134
6.3 The Choice of Python . . . 137
6.4 Example . . . 138
6.5 Design of the TMMElement Class . . . 141
6.5.1 Purpose/Overview . . . 141
6.5.2 Methods . . . 142
6.5.3 Properties . . . 142
6.5.4 Code/Usage . . . 143
6.5.5 Example: Torsional Spring/Damper . . . 148
6.6 Design of the TMMSystem Class . . . 150
6.6.1 Purpose/Overview . . . 150

6.6.2 Methods . . . 151
6.6.3 Properties . . . 151
6.6.4 Code . . . 151
6.6.5 An Example . . . 152
6.7 Bodeout Class . . . 155
6.8 TMMSubSystem Class . . . 156
6.9 Software Design for Symbolic Modeling . . . 157
VII SYSTEM IDENTIFICATION . . . 160
7.1 Introduction . . . 160
7.2 The Need . . . 160
7.3 Motivation . . . 160
7.4 Overview . . . 161
7.5 Goal . . . 162
7.6 Intelligent Automation . . . 163
7.7 Software Implementation . . . 164
7.7.1 An Example . . . 165
7.7.2 Automated Data Processing . . . 167
7.8 SAMII System ID Case Study . . . 167
7.8.1 Overview . . . 167
7.8.2 System Description . . . 167
7.8.3 Data Truncation . . . 169
7.8.4 General Approach . . . 171
7.8.5 Initial Attempt . . . 171
7.8.6 Quantifying the Accuracy of System ID Results . . . 174
7.8.7 Additional Approaches and Down-sampling . . . 178
7.8.8 Final Approach . . . 179
7.8.9 Conclusions . . . 185
7.9 Statistical Analysis of Bode Data . . . 190
7.9.1 Bias Error . . . 190
7.9.2 Random Errors . . . 191

VIIINUMERICAL CONCERNS **196**
8.1 Introduction 196
8.2 Floating-Point Accuracy 196
8.2.1 An Example 196
8.2.2 Floating-Point Subtraction 199
8.2.3 Floating-Point Analysis of the Pinned-Pinned Beam 199
8.2.4 A Pragmatic Approach 201
8.2.5 Symbolic Analysis 202
8.3 Repeated Roots 202
8.3.1 Overview 203
8.3.2 Illustrative Examples 204
IX CONCLUSIONS AND RECOMMENDATIONS **219**
9.1 Conclusions 219
9.2 Contributions 221
9.3 Recommendations for Future Work 222
APPENDIX A — DERIVATION OF A CONTINUOUS BEAM ELEMENT **224**
APPENDIX B — FEA CODE FOR L-SHAPED STRUCTURE **228**
APPENDIX C — PYTHON CODE **235**
REFERENCES **255**
VITA **260**

LIST OF TABLES

1 Comparison of the first 9 natural frequencies found by the TMM and FEA. 72

2 Crossover frequency and phase margin vs. K_θ. 83

3 Results from symbolic TMM control design. 102

4 Summary of the iteration process for the state-space design methodology of Book and Majette for the system in Figure 74. 104

5 Quantification of the accuracy of the initial system identification attempt. . 175

6 Quantitative comparison of all approaches to system identification. 179

7 Comparison of root finding results: symbolic ($\sinh\beta \sin\beta$) vs. numeric determinant ($|\mathbf{subU}|$). 201

8 Parameter values for SAMII's cantilever beam. 204

9 Table of the first two system eigenvalues for each case of EI_2/EI_1 considered (found using a numerical search algorithm). 207

10 The norm of each row of the $RREF(\mathbf{subU})$. 210

11 Eigenvalues of **subU** for each case of EI_2/EI_1 considered. 212

12 Singular values of **subU** for each case of EI_2/EI_1 considered. 213

13 Comparison of system eigenvalues from TMM and FEA for the nearly repeated and repeated root cases. 214

LIST OF FIGURES

1 Picture and schematic of SAMII: Small, Articulated Manipulator II. 2

2 A simple mass/spring system used to introduce the transfer matrix method (TMM). 15

3 Exploded view of the mass/spring system showing the state variables x_i and F_i. 15

4 Example system for TMM Bode analysis example. 18

5 Free body diagram of a mass to explain the negative sign in the forcing matrix of equation (23). 20

6 One of SAMII's hydraulic actuators, a rotary hydraulic motor controlled by a servo-valve 24

7 Comparison of actuator Bode plots θ/v from experimental data and from a model were the actuator is an angular velocity source with a first-order lag. 25

8 Picture of SAMII showing joint 2, the actuated joint for the experimental Bode plots in this thesis. 26

9 Block diagram of a hydraulic actuator model that assumes intrinsic velocity feedback. 27

10 Comparison of actuator Bode plots θ/v from experimental data and from a torque limiting FEA model. 27

11 Comparison of experimental and model Bode plots for an intrinsic velocity feedback model of a hydraulic actuator by Obergfell [47]. 28

12 SAMII in a configuration used for earlier investigations. 29

13 Set-up for testing in a vertical configuration. 29

14 Set-up for testing in a vertical configuration (as pictured) and a horizonal configuration (illustrated). 30

15 Bode plot of θ/M_z for the hydraulic actuator in horizonal and vertical configurations. 31

16 Bode plot of M_z/v for the hydraulic actuator in horizonal and vertical configurations. 31

17 Bode plot of θ/v for the hydraulic actuator in horizonal and vertical configurations. 33

18 Comparison of actuator Bode plots θ/v from experimental data and from a transfer matrix model where the actuator is an angular velocity source in series with a spring/damper element. 34

19 Overlay of the results from Figures 10 and 18, showing that the angular velocity source in series with a spring/damper element appears to better capture the dynamics than the torque limiting FEA model. 35

20 Picture of SAMII showing joint 2 and the two outputs: θ is the angular position of joint 2 and $\ddot{x}$ is the acceleration of SAMII's base (i.e. the end of the cantilever beam). 37

21 Block diagram of the system with position control (θ feedback) and vibration suppression ($\ddot{x}$ feedback). 38

22 Schematic of SAMII showing showing transfer matrix elements. 42

23 Block diagram of the open-loop system. 46

24 Open-loop actuator Bode plot θ/v comparing experimental data to the TMM model after curve fitting. 46

25 Open-loop flexible base Bode plot $\ddot{x}/v$ comparing experimental data to the TMM model after curve fitting. 47

26 A plot of the FFT's of $\ddot{x}$ and v which are used to calculate the Bode plot of Figure 25, along with the coherence between the two signals. 48

27 A plot of the FFT's of θ and v which are used to calculate the Bode plot of Figure 24, along with the coherence between the two signals. 48

28 Block diagram of the system with position control (θ feedback). 49

29 Closed-loop actuator Bode plot θ/θ_d for the system with θ feedback comparing experimental data to the transfer matrix model and a transfer function model. 50

30 Closed-loop flexible base Bode plot $\ddot{x}/\theta_d$ for the system with θ feedback comparing experimental data to the transfer matrix model and a transfer function model. 50

31 Block diagram for the system with θ and $\ddot{x}$ feedback (vibration suppression). Note that the θ feedback loop has been replaced by the equivalent closed-loop transfer function G_{cl}. 52

32 Closed-loop actuator Bode plot $\theta/\hat{\theta}_d$ for the system with θ and $\ddot{x}$ feedback (vibration suppression) comparing experimental data to the transfer matrix model and a transfer function model. 52

33 Closed-loop flexible base Bode plot $\ddot{x}/\hat{\theta}_d$ for the system with θ and $\ddot{x}$ feedback (vibration suppression) comparing experimental data to the transfer matrix model and a transfer function model. 53

34 This figure shows the effectiveness of the vibration suppression controller by comparing the experimental flexible base Bode response of a system with only θ feedback ($\ddot{x}/\theta_d$) to one with θ feedback and vibration suppression ($\ddot{x}/\hat{\theta}_d$). There is nearly a 15 dB reduction in amplitude of the response of the first mode of the structure. 54

35 Sketch of a 3D beam element labeling the axes. 56
36 Sketch of a beam undergoing positive angular deflection about the z-axis. . 57
37 Zooming in on Figure 36. 57
38 A differential beam element showing the relationship between curvature and strain. 58
39 A sketch showing the relationship between stress and moment on a differential element 58
40 Sketch of the forces and moments acting on a differential beam element bending about the z-axis (vibrating in the xy plane). 59
41 Free-body diagram for a rigid body rotating about and center of mass displacing along the x-axis 65
42 Free-body diagram for a rigid body rotating about the y-axis and center of mass displacing in the z direction. 65
43 Free-body diagram for a rigid body rotating about the z-axis and center of mass displacing in the y direction. 66
44 Comparison of TMM and FEA mode shape for mode 1: plotting all 6 states of the structure vs. the x coordinate of the mesh. 74
45 3D wireframe comparison of TMM and FEA mode shape for mode 1. . . . 74
46 Comparison of TMM and FEA mode shape for mode 2: plotting all 6 states of the structure vs. the x coordinate of the mesh. 75
47 3D wireframe comparison of TMM and FEA mode shape for mode 2. . . . 75
48 Comparison of TMM and FEA mode shape for mode 5: plotting all 6 states of the structure vs. the x coordinate of the mesh. 76
49 3D wireframe comparison of TMM and FEA mode shape for mode 5. . . . 76
50 Comparison of TMM and FEA mode shape for mode 6: plotting all 6 states of the structure vs. the x coordinate of the mesh. 77
51 3D wireframe comparison of TMM and FEA mode shape for mode 6. . . . 77
52 Comparison of TMM and FEA mode shape for mode 7: plotting all 6 states of the structure vs. the x coordinate of the mesh. 78
53 3D wireframe comparison of TMM and FEA mode shape for mode 7. . . . 78
54 Block diagram of the open-loop system. 80
55 Block diagram of the system with motion control (θ feedback) and vibration suppression ($\ddot{x}$ feedback). 80
56 Block diagram of the system with θ feedback. 81
57 Open-loop Bode plot for the actuator $G_p = \theta/v$. 82

58 Loop transfer Bode plots $G_\theta G_p$ for various values of K_θ. 82

59 Closed-loop actuator Bode plots G_{cl} for various values of K_θ. 83

60 Bode plot of the loop transfer function $G_\theta G_p$ for the optimal $K_\theta = 1.684$. . 84

61 Closed-loop actuator Bode plot $G_{\text{cl}} = \theta/\theta_d$ for the optimal $K_\theta = 1.684$. . . 85

62 Simplified block diagram with disturbance input. 85

63 Bode plot of $G_{\text{cl}}G_{\text{flexb}} = \ddot{x}/\theta_d$. 86

64 Closed-loop actuator Bode plots $\theta/\hat{\theta}_d$ for optimized first and second-order compensators. 89

65 Closed-loop flexible base Bode plots $\ddot{x}/\hat{\theta}_d$ for optimized first and second-order compensators. 89

66 Closed-loop disturbance rejection Bode plots $\ddot{x}/d$ for optimized first and second-order compensators. 90

67 Closed-loop actuator Bode plots $\theta/\hat{\theta}_d$ for sequential vs. simultaneous design of G_θ and G_a. 91

68 Closed-loop flexible base Bode plots $\ddot{x}/\hat{\theta}_d$ for sequential vs. simultaneous design of G_θ and G_a. 92

69 Disturbance rejection Bode plots $\ddot{x}/d$ for sequential vs. simultaneous design of G_θ and G_a. 92

70 Base acceleration $\ddot{x}$ response to a step in desired angle with and without acceleration feedback. 93

71 Base acceleration $\ddot{x}$ response to a step in desired angle with and without acceleration feedback (zooming in on the y-axis of Figure 70). 93

72 Base acceleration $\ddot{x}$ response to a step in desired angle with and without acceleration feedback (zooming in on the x-axis of Figure 70). 94

73 Sketch of a cantilever beam with a force at its free end 97

74 The two-flexible-link robot used by Book and Majette [9, 41] for TMM pole-placement control design. 100

75 Symbolic control design algorithm. 100

76 Block diagram of the pole-placement control problem for SAMII. 105

77 Open-loop and desired closed-loop pole-locations for the SAMII pole-placement design problem. 105

78 Adaptive interpolation algorithm moving from point A to point B. 110

79 Results from the pole-placement control design. 110

80 Comparison of actuator Bode plots $\theta/\hat{\theta}_d$ for the Bode and pole optimization control designs. 113

81 Comparison of flexible base Bode plots $\ddot{x}/\hat{\theta}_d$ for the Bode and pole optimization control designs. . . . 113

82 Comparison of disturbance rejection Bode plots $\ddot{x}/d$ for the Bode and pole optimization control designs. . . . 114

83 Loop transfer functions for G_θ design. . . . 116

84 Closed-loop actuator Bode plots θ/θ_d with $G_\theta = K_\theta\, G_{\text{cancel}}$. . . . 117

85 Closed-loop Bode plots for the actuator voltage v/θ_d with $G_\theta = K_\theta\, G_{\text{cancel}}$. 117

86 Bode plot for $\ddot{x}/\hat{\theta}_d$ investigating the robustness of the control design to changes in k_{base}. . . . 121

87 Bode plot for $\theta/\hat{\theta}_d$ investigating the robustness of the control design to changes in k_{base}. . . . 122

88 Bode plot for $\ddot{x}/\hat{\theta}_d$ investigating the robustness of the control design to changes in c_{base}. . . . 122

89 Bode plot for $\theta/\hat{\theta}_d$ investigating the robustness of the control design to changes in c_{base}. . . . 123

90 Bode plot for $\ddot{x}/\hat{\theta}_d$ investigating the robustness of the control design to changes in k_{joint1}. . . . 123

91 Bode plot for $\theta/\hat{\theta}_d$ investigating the robustness of the control design to changes in k_{joint1}. . . . 124

92 Bode plot for $\ddot{x}/\hat{\theta}_d$ investigating the robustness of the control design to changes in c_{joint1}. . . . 124

93 Bode plot for $\theta/\hat{\theta}_d$ investigating the robustness of the control design to changes in c_{joint1}. . . . 125

94 Bode plot for $\ddot{x}/\hat{\theta}_d$ investigating the robustness of the control design to changes in K_{act}. . . . 125

95 Bode plot for $\theta/\hat{\theta}_d$ investigating the robustness of the control design to changes in K_{act}. . . . 126

96 Bode plot for $\ddot{x}/\hat{\theta}_d$ investigating the robustness of the control design to changes in p_{act}. . . . 126

97 Bode plot for $\theta/\hat{\theta}_d$ investigating the robustness of the control design to changes in p_{act}. . . . 127

98 Bode plot for $\ddot{x}/\hat{\theta}_d$ investigating the robustness of the control design to changes in ks_{act}. . . . 127

99 Bode plot for $\theta/\hat{\theta}_d$ investigating the robustness of the control design to changes in ks_{act}. . . . 128

100 Bode plot for $\ddot{x}/\hat{\theta}_d$ investigating the robustness of the control design to changes in c_{act}. . . . 128

101 Bode plot for $\theta/\hat{\theta}_d$ investigating the robustness of the control design to changes in c_{act}. 129

102 Bode plot for $\ddot{x}/\hat{\theta}_d$ investigating the robustness of the control design to changes in $\text{Gain}_{\text{bode1}}$. 129

103 Bode plot for $\theta/\hat{\theta}_d$ investigating the robustness of the control design to changes in $\text{Gain}_{\text{bode1}}$. 130

104 Picture and schematic of SAMII used to illustrate the TMM model. 140

105 The absolute value of the characteristic determinant of a cantilever beam over a frequency range that includes the first two natural frequencies. . . . 153

106 A phase wrapping problem that needs to be corrected before system identification is attempted. 164

107 Experimental actuator Bode plot θ/v. 168

108 Experimental flexible base Bode plot $\ddot{x}/v$. 168

109 Actuator Bode plot θ/v with coherence showing the truncated range for initial system identification. 169

110 Flexible base Bode plot $\ddot{x}/v$ with coherence showing the truncated range for initial system identification. 170

111 Actuator Bode plot θ/v showing system ID results for an initial attempt based on curve-fitting the complex values for the transfer functions. 172

112 Flexible base Bode plot $\ddot{x}/v$ showing system ID results for an initial attempt based on curve-fitting the complex values for the transfer functions. 172

113 Linear magnitude ratio of the actuator Bode plot θ/v plotted with a linear x-axis. 173

114 Linear magnitude ratio of the flexible based Bode plot $\ddot{x}/v$ plotted with a linear x-axis. 173

115 Open-loop actuator Bode plot θ/v showing logarithmically spaced points along the frequency axis that will be used to quantify the accuracy of system identification. 175

116 Open-loop flexible base Bode plot $\ddot{x}/v$ showing logarithmically spaced points along the frequency axis that will be used to quantify the accuracy of system identification. 176

117 Closed-loop actuator Bode plot θ/θ_d (θ feedback) showing logarithmically spaced points along the frequency axis that will be used to quantify the accuracy of system identification. 176

118 Closed-loop flexible base Bode plot $\ddot{x}/\theta_d$ (θ feedback) showing logarithmically spaced points along the frequency axis that will be used to quantify the accuracy of system identification. 177

119 Closed-loop actuator Bode plot $\theta/\hat{\theta}_d$ (θ and $\ddot{x}$ feedback) showing logarithmically spaced points along the frequency axis that will be used to quantify the accuracy of system identification. 177

120 Closed-loop flexible base Bode plot $\ddot{x}/\hat{\theta}_d$ (θ and $\ddot{x}$ feedback) showing logarithmically spaced points along the frequency axis that will be used to quantify the accuracy of system identification. 178

121 Combined Bode and coherence plot for the actuator θ/v showing logarithmically spaced data points that will be used for system identification. 180

122 Combined Bode and coherence plot for the flexible base $\ddot{x}/v$ showing logarithmically spaced data points that will be used for system identification. . 181

123 Open-loop actuator Bode plot θ/v comparing curve-fitting results with and without logarithmic down-sampling of the data. 182

124 Open-loop flexible base Bode plot $\ddot{x}/v$ comparing curve-fitting results with and without logarithmic down-sampling of the data. 182

125 Closed-loop actuator Bode plot θ/θ_d with θ feedback comparing curve-fitting results with and without logarithmic down-sampling of the data. 183

126 Closed-loop flexible base Bode plot $\ddot{x}/\theta_d$ with θ feedback comparing curve-fitting results with and without logarithmic down-sampling of the data. . . 183

127 Closed-loop actuator Bode plot $\theta/\hat{\theta}_d$ with θ and $\ddot{x}$ feedback comparing curve-fitting results with and without logarithmic down-sampling of the data. . . 184

128 Closed-loop flexible base Bode plot $\ddot{x}/\hat{\theta}_d$ with θ and $\ddot{x}$ feedback comparing curve-fitting results with and without logarithmic down-sampling of the data. 184

129 Combined Bode and coherence plot for the actuator θ/v showing logarithmically spaced data points with more points per decade in regions near system resonances. 185

130 Combined Bode and coherence plot for the flexible base $\ddot{x}/v$ showing logarithmically spaced data points with more points per decade in regions near system resonances. 186

131 Open-loop actuator Bode plot θ/v comparing curve-fitting results from uniform and variable logarithmic down-sampling of the data. 187

132 Open-loop flexible base Bode plot $\ddot{x}/v$ comparing curve-fitting results from uniform and variable logarithmic down-sampling of the data. 187

133 Closed-loop actuator Bode plot θ/θ_d with θ feedback comparing curve-fitting results from uniform and variable logarithmic down-sampling of the data. . 188

134 Closed-loop flexible base Bode plot $\ddot{x}/\theta_d$ with θ feedback comparing curve-fitting results from uniform and variable logarithmic down-sampling of the data. 188

135 Closed-loop actuator Bode plot $\theta/\hat{\theta}_d$ with θ and $\ddot{x}$ feedback comparing curve-fitting results from uniform and variable logarithmic down-sampling of the data. 189

136 Closed-loop flexible base Bode plot $\ddot{x}/\hat{\theta}_d$ with θ and $\ddot{x}$ feedback comparing curve-fitting results from uniform and variable logarithmic down-sampling of the data. 189

137 An illustration of bias error for the Bode plot of a lightly damped system. . 190

138 Bode plot of SAMII's flexible base $\ddot{x}/v$ with varying T. 191

139 Bias error ε_B for $\ddot{x}/v$ with $T = 40$ seconds. 192

140 Bias error ε_B for $\ddot{x}/v$ with $T = 40$ seconds (zooming in on Figure 139). . . . 192

141 Bode plot with coherence for $\ddot{x}/v$ with 3 tests of $T = 100$ seconds and averaging across 5 frequency steps for $f > 2.1$ Hz. 194

142 Random error ε_R corresponding to the Bode plot of Figure 141. 194

143 Bode plot with coherence for $\ddot{x}/v$ showing the 95% confidence interval. . . . 195

144 Value of the characteristic determinant from numeric evaluation based on the matrix in equation (241) and from the symbolic expression in equation (243). 200

145 Plot of the characteristic determinant near the first two system eigenvalues for the cases $EI_2/EI_1 = 1.5$, 1.1, 1.01, 1.001, and 1.0 205

146 Zooming in on Figure 145 to better show the case $EI_2/EI_1 = 1.01$ 206

147 Zooming in further on Figure 146 to better show the cases $EI_2/EI_1 = 1.001$ and 1.0 206

148 Comparison of mode shapes from TMM and FEA for mode 1 for the case $EI_2/EI_1 = 1.001$. 215

149 3D visualization of mode shapes from TMM and FEA for mode 1 for the case $EI_2/EI_1 = 1.001$. 215

150 Comparison of mode shapes from TMM and FEA for mode 2 for the case $EI_2/EI_1 = 1.001$. 216

151 3D visualization of mode shapes from TMM and FEA for mode 2 for the case $EI_2/EI_1 = 1.001$. 216

152 Comparison of mode shapes from TMM and FEA for mode 1 for the case $EI_2/EI_1 = 1.0$. 217

153 3D visualization of mode shapes from TMM and FEA for mode 1 for the case $EI_2/EI_1 = 1.0$. 217

154 Comparison of mode shapes from TMM and FEA for mode 2 for the case $EI_2/EI_1 = 1.0$. 218

155 3D visualization of mode shapes from TMM and FEA for mode 2 for the case $EI_2/EI_1 = 1.0$. 218

SUMMARY

Control design for flexible robots is a challenging problem. Part of the difficulty comes from a lack of controls-focused modeling tools. Practical flexible robots have several aspects that make them difficult to model: continuous elements, complicated actuators, multiple feedback loops, non-collocated sensors and actuators, and the ability to take on arbitrary three-dimensional poses. Even if existing techniques for modeling flexible structures could model the closed-loop response of a hydraulically-actuated flexible robot with a vibration suppression controller, how would such a model be used for control design?

This work presents the development of a modeling approach that meets the needs of a controls engineer. The approach is based on the transfer matrix method (TMM). The TMM has been expanded in several ways to enable it to accurately model practical flexible robots. Quantitative agreement is shown between model and experiment for the interaction of a hydraulic actuator and a flexible structure as well as for the closed-loop response of a system with vibration suppression.

Once the ability to model the closed-loop response of the system has been demonstrated, this work focuses on using the model for control design. Control design is facilitated by symbolic implementation of the TMM, which allows closed-form expressions for the closed-loop response of the system to be found without discretization. These closed-form expressions will be transcendental transfer functions for systems with continuous elements. These transfer functions can then be used in various optimization approaches for designing the closed-loop system response.

CHAPTER I

INTRODUCTION

1.1 Background/Problem Statement

Many researchers and engineers over the years have claimed that flexible robots can be lighter, faster, and cheaper to actuate than similar rigid robots. These benefits may be especially important in aerospace applications, where structures must be lightweight. But in order to realize these benefits, some means of dealing with deflection is required. In the case of some long-reach robots, flexibility may be practically unavoidable. Controls and robotics engineers need tools to design vibration suppression schemes or command shaping algorithms in order to reap the potential benefits of using flexible robots in practical applications.

Modeling and control design of flexible robots with realistic actuators is a challenging problem. A flexible robot will likely consist of distributed and lumped parameter elements. There will likely be many of these links connected to one another. Practical robots may have complicated actuators that interact with the robot structure. Vibration suppression schemes may lead to multiple control loops where sensors and actuators are not strictly collocated. The robot's dynamics may change considerably based on the pose of the robot. The pose of a robot refers to the spatial orientation of links with respect to one another or some reference frame. Modeling and control design tools are needed that can handle all of these possibilities.

Modeling and control of flexible robots is a research area that is full of interesting challenges. Hopefully the knowledge and techniques coming out of this work will benefit the wider community of people working in the more general area of active vibration suppression.

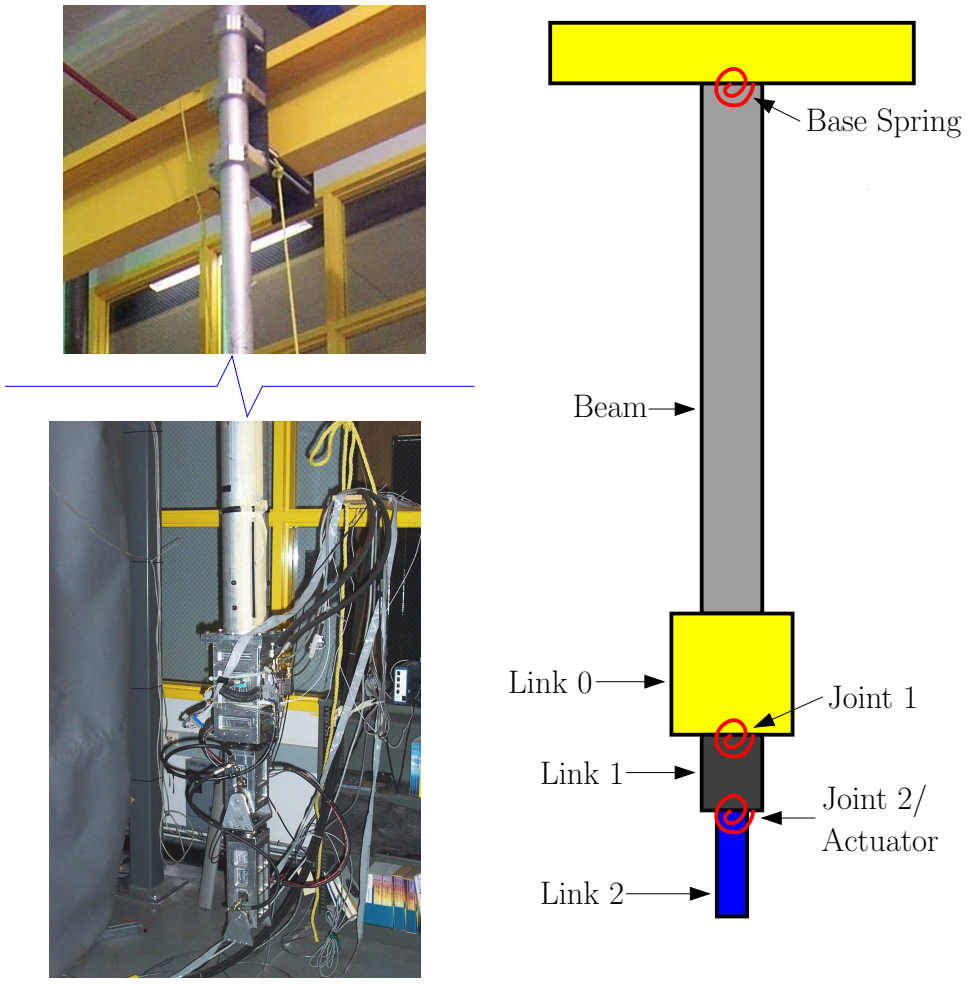

Figure 1: Picture and schematic of SAMII: Small, Articulated Manipulator II.

1.1.1 Problem Statement

The primary experimental test bed for this work is a robot known as SAMII: Small, Articulated Manipulator II. SAMII is a small robot with rigid links mounted on the end of a 5 meter long cantilever beam as shown in Figure 1.

Earlier investigations on SAMII highlighted the need for a good model of the system that could help interpret experimental results and facilitate control design. But existing modeling approaches seemed lacking, especially from the control design viewpoint. Modeling feedback using many FEA packages is not straightforward and the assumed modes method is difficult to implement on robots with many links. The primary focus of this thesis is to

meet this need for a better tool for modeling and control design of flexible robots/structures. This tool will be used for motion control and vibration suppression design.

This modeling technique should have the following properties:

- Modeling feedback must be straightforward
- Modular
- Outputs mode shapes and natural frequencies of the system as a whole
- Straightforward handling of element connectivity conditions (the boundary conditions between model elements/robot links)
- Capable of modeling arbitrary robot configurations
- Accurate modeling of complex actuators
- Useful for control design

While the aim is to provide a tool that can be used on a broad range of control design problems, at a minimum the technique must apply to SAMII. SAMII has at least four important properties to model:

1. Continuous elements
2. Hydraulic actuators
3. Three-dimensional poses
4. Non-collocated feedback

1.1.2 An Assertion

An underlying assertion of this work is that a new modeling and control design tool is needed. While several techniques for modeling flexible structures exist, they are not ideal for control design. Finite element analysis (FEA) is a prevalent approach, but learning commercial packages can be cumbersome and finding one that facilitates modeling feedback

control is not easy. Combining that with the need to model hydraulic actuators and multi-body dynamics makes this a daunting task for a controls engineer who is not an expert in FEA.

Even if a particular FEA package is capable of modeling the closed-loop control response of a system, using such a model for control design may still be cumbersome. The best one could hope for in using FEA for control design of flexible structures would be an iterative approach analogous to the method of Book and Majette [9, 41], where the FEA analysis outputs a state-space model based on modal discretization. The state-space model is used for control design and the controller is substituted back into the FEA model. Iteration will likely be required because the feedback control will alter the mode shapes of the system, affecting the modal discretization.

The assumed modes method (AMM) is another approach that has been used to model flexible robots. However, the difficulty of using the method grows considerably as the number of serially connected bodies grows. Many researchers who use the AMM on serial robots approximate the element connectivity conditions and the effects of distal links because of this difficulty.

1.1.3 Claim

This thesis seeks to demonstrate that the transfer matrix method can be the right tool for modeling and control design of flexible robots/structures. It is modular, so modeling robots made up of many links is straightforward. It is useful to the controls engineer: feedback can be modeled; Bode plots are output naturally; and transfer functions can easily be incorporated into transfer matrices. Element connectivity conditions are modeled exactly and automatically. One interesting theoretical capability of the method is that continuous elements can be modeled without discretization.

The primary limitation of the method is that it is inherently linear.

1.1.4 Contributions

The contributions of this work fall into three main categories: expanding the ability of the TMM to model practical flexible robots, enabling the TMM to be used for control design,

and designing software for TMM analysis.

1.1.4.1 Modeling

The contributions in the area of modeling allow the TMM to be applied to a broader range of problems. There are three main areas of modeling contributions: hydraulic actuators, three-dimensional poses, and non-collocated feedback.

A significant amount of time and effort was spent developing a hydraulic actuator model that accurately captures the interaction of the actuator with the structure at resonance. In the end, the actuator model is pretty simple, but it is different from any seen in the literature. It was also placed in transfer matrix form and experimentally validated to capture the essential dynamics.

This thesis contributes transfer matrices for modeling arbitrary 3D poses of flexible robots or other actuated structures. While three-dimensional modeling using the TMM has been done before, the matrices derived in this thesis have been numerically validated by comparison with FEA. This ensures that all of the details associated with the derivation of 3D transfer matrix elements have been handled correctly.

While non-collocated feedback can be dangerous from the stand point of system stability, it is practically unavoidable in some situations. SAMII has two control loops where it would be very difficult and/or expensive to precisely collocate the actuators and sensors. This work removes a limitation of the TMM by presenting a method for modeling non-collocated feedback.

1.1.4.2 Control Design

Contributions in control design include symbolic implementation of the transfer matrix method, creation of a control design technique based on simultaneous optimization of multiple Bode plots, and algorithm development for placing poles of systems with continuous elements. Symbolic implementation of the TMM enables control design by allowing the method to output closed-form expressions for the closed-loop response of the system. For systems with continuous elements, these expressions will involve transcendental equations

whose roots are manipulated by proper choices of control parameters. Algorithms for reliably finding and optimizing these closed-loop poles have been created. A method for control design based on simultaneous optimization of multiple Bode plots is also presented.

1.1.4.3 Software Design

As part of this work, a software package has been created for TMM analysis. This software makes the power of the TMM available to controls or vibrations engineers who are not experts in the TMM. The software is object oriented and designed to be user extensible and customizable. Classes have been created for transfer matrix elements and transfer matrix systems. These classes are useful software abstractions and form a basis for user extensibility through inheritance. The software also includes integrated system identification capabilities.

1.2 Literature Review

The literature review is divided into three areas: modeling of flexible robots, control of flexible robots, and the progression of work at Georgia Tech.

1.2.1 Modeling of Flexible Robots

Accurate modeling of flexible robots requires that the distributed nature of the flexible links be considered. Such an approach leads to partial differential equations instead of ordinary ones. This can make modeling very challenging especially when these distributed parameter links are connected to rigid links and complicated actuators (SAMII's actuators are all hydraulic). Several approaches are common in the literature including the assumed modes method (AMM), finite element analysis (FEA), and approaches like the transfer matrix method (TMM) that deal with the partial differential equations without discretization.

1.2.1.1 Assumed Modes Method

Book [7] proposed an efficient method for incorporating link flexibility into a recursive Lagrangian dynamic model using 4×4 transformation matrices to capture the flexible link deformation.

De Luca and Siciliano [20] presented a method for determining a closed-form model of a flexible robot undergoing small vibrations based on the assumed modes method. The equations of motion for a two-link robot were explicitly determined. Each link was modeled as clamped at its proximal end and having the mass and second moment of inertia of the subsequent links at its distal end. The configuration dependence of the second moment of inertia of the subsequent links created a time-varying boundary condition. The effects of approximating this time-varying boundary condition with a constant were discussed and extensive simulation results were presented.

Chalhoub and Chen [19] used floating reference frames to extend the work in [7] to include systems with prismatic joints, torsional and out-of-plane vibrations, and geometric foreshortening of the links. The work in [19] used structural flexibility matrices to give fully general homogeneous transformation matrices for flexible links, including the possibility of large rotations.

Subudhi and Morris [56] used an Euler-Lagrange formulation and the assumed modes method in a systematic modeling technique for manipulators with flexible joints and flexible links.

Kang and Mills [29] used Lagrange multipliers to extend the assumed modes method to modeling of parallel manipulators.

Bascetta and Rocco [4] combined screw theory with the assumed modes method in an attempt to develop a computationally efficient model for a flexible manipulator with motors at the joints. The gyroscopic effects of the motors were included in this work so that the dynamic effects of the motors were fully modeled.

1.2.1.2 Finite Element Modeling

Brüls et al. [13, 15, 14] have done extensive work in the area of nonlinear FEA modeling of robots and mechatronic systems. Their work has been specifically applied to SAMII [31] and RALF. RALF stands for Robot Arm, Large and Flexible. RALF is a robot with two flexible links. Their approach was based on developing a full nonlinear model of the robot and linearizing about various points in the configuration space. Techniques for moving

between the linearized models as the robot undergoes large motions have been developed.

Tokhi et al. [57] used the finite element method to model a one link flexible manipulator with a payload at the end. The work in [57] showed good agreement between model and experiment for the simple system considered; the system was essentially a slewing beam, with only one link confined to planar rotation about a proximal joint. Although the system considered was fairly simple, the work demonstrated the ability of FEA to model flexible robots.

Zhou et al. [67] presented a method based on component mode synthesis to use FEA to model a flexible payload that was being manipulated by a rigid robot. An example system was presented that included a rigid robot handling sheet metal for automated automobile manufacturing. Component mode synthesis allowed the modeler to handle the constraints on the flexible payload at its attachments to the rigid robot in a straight forward manner.

Nagaraj et al. [45] combined FEA with a momentum balance technique to predict the vibrations induced by the locking of a joint during the deployment of a solar array on a flexible spacecraft. This approach may be applicable to modeling the vibrations resulting from rapid closure of a hydraulic valve.

Various papers have been written on addressing modeling issues specific to FEA, i.e. problems that only arise because the finite element method was being used. Megahed and Hamza [43] focused on issues involved in modeling flexible links with rigid attachments that extended into the link length. Torby and Kimura [58] discussed problems with individual elements becoming excessively stiff during the motion of prismatic joints. If the portion of an element that was sticking out beyond a prismatic joint was too short, the stiffness of the element could increase to where it caused numerical issues with the model. If any portion of the system oscillates faster than the FEA sample time, numeric instability could result.

1.2.1.3 Transfer Matrix Method

The transfer matrix method was introduced by Pestel and Leckie [49]. Book [10] specifically applied the method to design, modeling, and control of flexible robots. He also expanded the method by contributing transfer matrices for rigid links and actuated joints (with transfer

functions of joint angle vs. torque).

Book et al. [11, 12] used the TMM to analyze the Remote Manipulator System on the space shuttle, along with its flexible payloads. They developed a FORTRAN software package for TMM analysis called the Distributed Systems Analysis Package (DSAP). Their software included the ability to model systems with parallel links and multiple branches.

Yu et al. [64] derived a Transfer Dynamic Stiffness Matrix (TDSM), based on the transfer matrix method and used this TDSM in the analysis of flexible space structures. The TDSM was frequency dependent, much like a transfer matrix, but it related force and displacement like a regular stiffness matrix rather than transferring states from one end of an element to the other. Much like the transfer matrix method, their approach enabled exact analysis of structures while requiring much fewer states than FEA.

1.2.1.4 Dynamic Stiffness Method

The dynamic stiffness method is closely related to the transfer matrix method. It also models dynamic systems without discretization [24]. Banerjee [2] gave an overview of the method and cited many instances where the method has been applied to beams and other aerospace structures. Banerjee and Fisher [3] applied the method to an axially loaded beam element and derived a dynamic stiffness matrix for coupled bending and torsional deformation.

The dynamic stiffness method differs from the transfer matrix method in that it is based on arranging the differential equations for the system into matrices relating forces and moments to displacements and rotations rather than transferring the states from one end of an element to the next. For example, a dynamic stiffness matrix model could take the form

$$\mathbf{F} = \mathbf{K}\boldsymbol{\delta} \tag{1}$$

where $\mathbf{K}$ is the dynamic stiffness matrix, $\mathbf{F}$ will include forces and moments from both ends of the element and $\boldsymbol{\delta}$ will include displacements and rotations from both ends of the element. In contrast, a transfer matrix would transfer matrix model takes the form

$$\mathbf{z}_{\mathrm{i}} = \mathbf{U}\,\mathbf{z}_{\mathrm{i}-1} \tag{2}$$

where $\mathbf{U}$ is the transfer matrix, $\mathbf{z}_i$ will include all states (force, moment, displacement, and rotation) at one end of the element and $\mathbf{z}_{i-1}$ will include all the states at the other end of the element.

The transfer matrix representation makes the assembling of the system model easier when working with serial connections of links. When dealing with closed-kinematic chains or other more complicated topologies, the dynamic stiffness method may be preferable. The assembling of the global model for the dynamic stiffness method closely resembles assembling the global model for the finite element method.

1.2.1.5 Transfer Functions of Continuous Elements

Yang [60, 61] presented a related approach to finding exact, closed-form transfer functions for systems involving continuous elements. Much like the transfer matrix method and the dynamic stiffness method, the approach was based on taking the Laplace transform of the partial differential equations of the system. Yang's approach was based on Green's functions and can handle non-self-adjoint systems and inhomogeneous boundary conditions.

1.2.1.6 Comparison of Modeling Techniques

Piedboeuf and Moore [50] compared six different modeling techniques for modeling a flexible beam rotating about a joint at the proximal end. The methods compared included 3 continuous methods (Hamilton's principle, Lagrange's equations for quasi-coordinates, and Newton-Euler equations) and three discrete methods (Lagrange's equations, Jourdain's principle, and Newton-Euler equations). The main conclusion was that if second order effects related to axial foreshortening were ignored, the different methods produced models that were different from one another and different from the models that included the second order effects.

1.2.2 Controls

Many approaches to controls exist in the literature. Many of these have been applied to the control of flexible robots. The application of state-space based pole-placement techniques to flexible systems was discussed in [8]. Such techniques should be able to move lightly

damped poles to locations with higher damping. However, initial experiments have shown that this approach drives higher modes of SAMII unstable.

Kwon and Book [32] developed a method for determining the feedforward torque required for end-point position tracking without exciting structural vibrations for a single-link flexible manipulator. The approach was based on dividing the inverse dynamic system into causal and anti-causal parts. The inverse dynamics analysis outputs the feedforward torque along with trajectories for all state variables. Insight into the non-causal nature of the inverse dynamics problem was given. The feedforward torque control was combined with a feedback controller and a friction compensator to produce excellent agreement with experimental results for this approach.

Siciliano and Book [54] developed a singular perturbation approach to the control of flexible-link manipulators. This approach provided an additional means of model reduction by allowing the system to be broken into slow and fast subsystems. For the case of flexible manipulators, the slow subsystem refers to the rigid-body motion. The fast subsystem refers to vibrations about the nominal motion of the slow subsystem. Rigid robot control techniques were applied to the slow subsystem, and then a stabilizing controller was developed for the fast subsystem. Simulation results demonstrated that the singular perturbation controller outperforms a typical state-feedback regulator. Rocco and Book [52] extended this work to controlling the position of a flexible robot as well as its contact force with a rigid environment. Coordinate partitioning was used to reduce the order of the model in accounting for the constraints of the rigid environment. The derivation preserved the separation of the slow and fast systems while accounting for these constraints. Simulation results were presented for RALF.

Calise et al. [16] presented an approach for designing optimal compensators for the slow and fast subsystems of two-time scale systems. Care was taken to ensure that fast subsystem spillover does not destabilize the slow subsystem by requiring that the fast compensator have zero magnitude at low frequencies. This requirement was shown not to affect the vibration suppression capabilities of the fast subsystem because the flexible dynamics must be at higher frequencies by virtue of the two-time scale assumption.

Luo [38] presented a rigorous proof of the ability of direct strain feedback control to damp vibrations for a single-flexible-link robot. The proof was based on the concept of A-dependent operators in Hilbert spaces. The theoretical results were experimentally verified. Vibration damping controllers were presented for both torque and speed-controlled DC motors. The torque control formulation requires the feedback of strain rate, which is difficult to measure in practice. For motors where the input signal is proportional to rotational speed, the strain signal can be feedback directly. Experimental results were shown for this case. The formulation used the partial differential equations directly without discretization.

An augmenting adaptive controller based on neural networks has been applied to several flexible systems and to SAMII in particular by Yang, Calise, and Craig [17, 62, 63]. The approach has been shown to be effective in compensating for parametric errors, unmodeled dynamics, external disturbances (including unmatched/non-collocated), and actuator nonlinearities. An external model-following control was developed that does not rely on inversion and can therefore be applied to non-minimum phase systems.

Morris and Madani [44] compared an open-loop computed torque controller to a quadratic optimal controller for a two-flexible-link robot and found that the optimal controller performs better. Lahdhiri and Elmaraghy [34] also applied optimal control techniques to the control of flexible robots. The approach was based on the combination of feedback linearization and linear-quadratic-Gaussian/loop-transfer-recovery techniques. Experimental results showed excellent tracking in the presence of measurement noise.

Other approaches that seem applicable to flexible robots include robust control, H_∞, and sliding mode control. Sliding mode control has been applied to flexible structures by Kao and Sinha [30]. They used the input identifiable form to transform the construction of the sliding manifold to a pole-placement problem. Simulations were performed for a cantilever beam and a truss structure. Robustness of the controllers was tested and the truss controller was modified due to a lack of robustness.

Hisseine and Lohmann [26] used both sliding mode and H_∞ control in the design of a controller for a one link flexible robot. Experiments were performed to verify robustness to changes in tip mass. Vibration suppression was not explicitly considered.

Xu and Cao [59] applied sliding mode control to a single-flexible-link robot. The approach used a reference model that has the same number of poles as the physical system, but those poles were placed on the real axis. The reference model also had no zeros. This reference model was used to define the switching surface. Simulation results showed effective vibration suppression: the system responded like the reference model while under sliding mode control.

Residual mode filtering (RMF) [51] is an approach that seems particularly relevant because it deals explicitly with eliminating higher mode instability. The gains used to control lower modes are not restricted by higher mode instability. An observer is used to subtract the higher mode content from the feedback signals, so that the higher modes are left at their open-loop pole locations.

Sano [53] used residual mode filtering to guarantee the stability of a finite dimensional H_∞ controller on an infinite dimensional structure.

1.2.3 Progression of the Work at Georgia Tech

The Intelligent Machine Dynamics Laboratory (IMDL) at Georgia Tech has been active in modeling and control of flexible robots for some time. Many diverse approaches to predicting and reducing vibrations have been taken.

Hastings and Book [25] used strain gauges to reconstruct modal amplitudes of a single flexible link. These amplitudes were then used in full-state feedback control.

Alberts [1] demonstrated the effectiveness of using passive damping to augment active control of flexible manipulators. A model for the beam with a distributed damping layer was derived. Simulations showed that passive damping reduced the system's susceptibility to spillover effects.

Huggins [27] performed extensive modeling of RALF. An assumed modes model using two modes per link was compared to a finite element model. Modal analysis was also performed and used to refine the finite element model. As the FEA model was made more and more realistic, it deviated from the assumed modes model, but agreed well with the experimental results. The FEA model was also used to guide the modal analysis especially

at higher frequencies were the modes were not well separated.

Yuan et al. [66] developed a model reference adaptive controller for a single link flexible arm. Simulations with varying payloads demonstrated the superior robustness of this controller compared to non-adaptive optimal control.

Lew and Book [36] investigated bracing a flexible manipulator against its environment to reduce vibrations. A hybrid controller was developed to control both end-point position and contact force at the bracing points.

Yuan et al. [65] extended the work in [7] to include kinematic constraints that prevent a link from rotating based on deflection of the preceding link. This method was applied to two experimental test beds at Georgia Tech. Experimental and finite element mode shapes were used in the assumed modes model.

Lee [35] developed an analytical model for RALF based on the assumed modes method. Component mode synthesis was used to determine the assumed mode shapes and this model was compared to FEA and experiments.

Magee [40] developed and patented the Optimal Arbitrary Time-Delay Filter (OATF) for command shaping. The robustness of the OATF was investigated through testing on RALF.

Cannon [18] designed and built SAMII. He also developed a control scheme that combined command shaping and inertial damping to reduce vibrations in SAMII. His work used one degree of freedom of SAMII to suppress vibration of one mode of the base.

Loper [37] extended the work of Cannon [18] to include two-dimensional (2D) vibration suppression using two degrees of freedom. His work also included using inverse dynamics to calculate the interaction force between SAMII and the flexible base.

Obergfell [48] performed experimental identification of RALF. This work was combined with a substantial effort toward sensing the end-point position of RALF's flexible links and feeding back these end point position measurements in a control scheme.

George [23] expanded on the work of Loper [37] and Cannon [18] by examining a six degree of freedom vibration problem. She developed a three degree of freedom vibration suppression controller. Her work also included an investigation of inertia singularities within

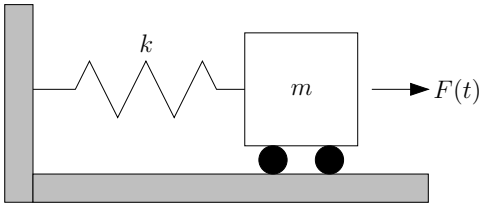

Figure 2: A simple mass/spring system used to introduce the transfer matrix method (TMM).

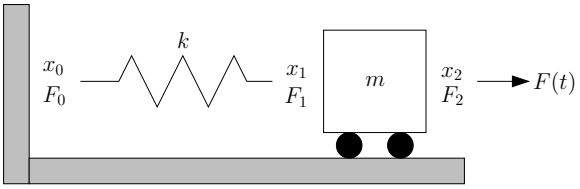

Figure 3: Exploded view of the mass/spring system showing the state variables x_i and F_i.

the workspace. This investigation resulted in a quantitative performance index that predicts the effectiveness of the vibration suppression system for any rigid robot configuration.

1.3 Introduction to the Transfer Matrix Method (TMM)

The transfer matrix method is an approach to system modeling that breaks a system into components whose dynamic properties can be expressed in matrix form. These matrices transfer a state vector from one end of an element to the other. The method is explained in detail by Pestel and Leckie [49].

Consider as a simple example the mass/spring system shown in Figure 2. The system is shown in an exploded view with the state variables x_i and F_i in Figure 3. Because the spring is massless,

$$F_1 = F_0 \tag{3}$$

and the displacement across the spring will be

$$x_1 = \frac{F_0}{k} + x_0. \tag{4}$$

Equations 3 and 4 could be written in matrix form as

$$\begin{Bmatrix} x_1 \\ F_1 \end{Bmatrix} = \begin{bmatrix} 1 & 1/k \\ 0 & 1 \end{bmatrix} \begin{Bmatrix} x_0 \\ F_0 \end{Bmatrix} \tag{5}$$

The column vectors are the state vectors at the left and right end of the spring, respectively, and the square matrix is the transfer matrix for a spring element. This matrix transfers the state from the left end of the spring (location 0) to the right end of the spring (location 1).

Focusing on the mass, since it is rigid,

$$x_2 = x_1 \tag{6}$$

and Newton's second law gives

$$F_2 = F_1 + ms^2 x_1 \tag{7}$$

Equations 6 and 7 can be written in matrix form as

$$\begin{Bmatrix} x_2 \\ F_2 \end{Bmatrix} = \begin{bmatrix} 1 & 0 \\ ms^2 & 1 \end{bmatrix} \begin{Bmatrix} x_1 \\ F_1 \end{Bmatrix} \tag{8}$$

Substituting for x_1 and F_1 from equation (5) into equation (8) gives

$$\begin{Bmatrix} x_2 \\ F_2 \end{Bmatrix} = \begin{bmatrix} 1 & 0 \\ ms^2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1/k \\ 0 & 1 \end{bmatrix} \begin{Bmatrix} x_0 \\ F_0 \end{Bmatrix} \tag{9}$$

or

$$\begin{Bmatrix} x_2 \\ F_2 \end{Bmatrix} = \begin{bmatrix} 1 & 1/k \\ ms^2 & ms^2/k + 1 \end{bmatrix} \begin{Bmatrix} x_0 \\ F_0 \end{Bmatrix} \tag{10}$$

For the free response of the system, the boundary conditions are

$$x_0 = 0 \tag{11}$$

$$F_2 = 0 \tag{12}$$

Substituting the boundary conditions into equation (10) gives

$$\begin{Bmatrix} x_2 \\ 0 \end{Bmatrix} = \begin{bmatrix} 1 & 1/k \\ ms^2 & ms^2/k + 1 \end{bmatrix} \begin{Bmatrix} 0 \\ F_0 \end{Bmatrix} \tag{13}$$

The bottom row of this equation gives

$$\left(\frac{ms^2}{k} + 1 \right) F_0 = 0 \tag{14}$$

which means that for a non-trivial solution

$$\frac{ms^2}{k} + 1 = 0 \tag{15}$$

or

$$s^2 = -\frac{k}{m} \quad \text{or} \quad s = \pm j\sqrt{\frac{k}{m}} \tag{16}$$

The imaginary part of s is the natural frequency of the system.

The forced response of the system can be found by substituting $x_0 = 0$ and $F_2 = F$ into equation (10):

$$\begin{Bmatrix} x_2 \\ F \end{Bmatrix} = \begin{bmatrix} 1 & 1/k \\ ms^2 & ms^2/k + 1 \end{bmatrix} \begin{Bmatrix} 0 \\ F_0 \end{Bmatrix} \tag{17}$$

The bottom row of equation (17) can be solved for the transfer function

$$\frac{F_0}{F} = \frac{k}{ms^2 + k} \tag{18}$$

Substituting this result for F_0 in the top row gives

$$\frac{x_2}{F} = \frac{1}{ms^2 + k} \tag{19}$$

Obviously, there are easier ways to get these same results using other approaches for this very simple system. But for more complicated systems, a modular approach where each piece can be described using a matrix and the system model is simply found by multiplying element matrices is very useful.

Part of the power of the TMM lies in its ability to handle distributed parameter systems. It can model them without discretization and can mix distributed and lumped parameter elements easily. The derivation of a continuous beam element is discussed in section 4.2.1 and appendix A.

1.3.1 TMM Mode Shape Determination

An explanation of how to find system mode shapes using the TMM is given in section 4.5.1.

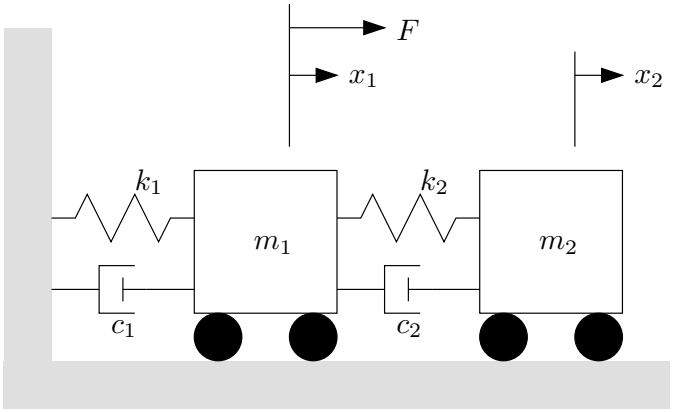

Figure 4: Example system for TMM Bode analysis example.

1.3.2 Bode Analysis

As a simple example of how to use the TMM to analyze the forced response of a system, consider the two mass system of Figure 4. When the actuation does not occur on the last element, a means of injecting forces, moments, or displacements into the middle of the TMM model is needed. This is accomplished through augmented transfer matrices. An augmented mass transfer matrix will take the form

$$\mathbf{U}_{\mathrm{m}} = \left[\begin{array}{cc|c} 1 & 0 & 0 \\ m\,s^2 & 1 & 0 \\ \hline 0 & 0 & 1 \end{array}\right] \tag{20}$$

An extra row and column have been added where all the elements are 0 except the lower right one.

An augmented spring/damper matrix will be

$$\mathbf{U}_{\mathrm{sd}} = \left[\begin{array}{cc|c} 1 & \dfrac{1}{c\,s+k} & 0 \\ 0 & 1 & 0 \\ \hline 0 & 0 & 1 \end{array}\right] \tag{21}$$

where the state vector is

$$\mathbf{x} = \left\{\begin{array}{c} x \\ F \\ 1 \end{array}\right\} \tag{22}$$

These augmented transfer matrices allow the force on mass 1 to be injected with a forcing matrix of the form

$$\mathbf{U}_{\mathrm{F}} = \left[\begin{array}{cc|c} 1 & 0 & 0 \\ 0 & 1 & -F \\ \hline 0 & 0 & 1 \end{array}\right] \tag{23}$$

This matrix will be used with the state vector of equation (22) to calculate the change of states across a forcing element according to

$$\mathbf{z}_{\mathrm{i}} = \mathbf{U}_{\mathrm{F}}\, \mathbf{z}_{\mathrm{i-1}} \tag{24}$$

The first row of this equation says the forcing element does not change the deflection:

$$x_i = x_{i-1} \tag{25}$$

The second row models the injection of the external force into the model:

$$F_i = F_{i-1} - F \tag{26}$$

The negative sign on the external force may be counter-intuitive considering that the force is in the positive direction. To understand this sign, consider the free body diagram of a mass as shown in Figure 5. Summing the forces on this mass gives

$$ms^2x = F_i + F - F_{i-1} \tag{27}$$

Transfer matrix analysis is focused on deriving matrices that calculate the states at station i based on the states at station $i-1$. Solving equation (27) for F_i gives

$$F_i = ms^2x + F_{i-1} - F \tag{28}$$

The mass transfer matrix takes care of the terms $ms^2x + F_{i-1}$, leaving the forcing matrix to handle the term $-F$.

1.3.3 System Model

The system transfer matrix model will be

$$\mathbf{z}_{\mathrm{end}} = \mathbf{U}_{\mathrm{sys}}\, \mathbf{z}_{\mathrm{base}} \tag{29}$$

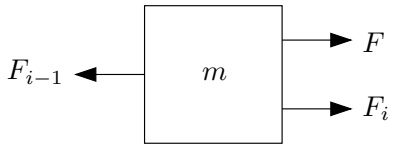

Figure 5: Free body diagram of a mass to explain the negative sign in the forcing matrix of equation (23).

where

$$\mathbf{U}_{\text{sys}} = \mathbf{U}_{\text{m2}}\,\mathbf{U}_{\text{sd2}}\,\mathbf{U}_{\text{F}}\,\mathbf{U}_{\text{m1}}\,\mathbf{U}_{\text{sd1}} \tag{30}$$

$\mathbf{U}_{\text{m1}}$ and $\mathbf{U}_{\text{m2}}$ are mass transfer matrices of the form given in equation (20). $\mathbf{U}_{\text{sd1}}$ and $\mathbf{U}_{\text{sd2}}$ are spring/damper transfer matrices of the form given in equation (21). The state vectors are

$$\mathbf{z}_{\text{end}} = \begin{Bmatrix} x_{\text{end}} \\ F_{\text{end}} \\ 1 \end{Bmatrix} \tag{31}$$

and

$$\mathbf{z}_{\text{base}} = \begin{Bmatrix} x_{\text{base}} \\ F_{\text{base}} \\ 1 \end{Bmatrix} \tag{32}$$

The system boundary conditions state that

$$x_{\text{base}} = 0 \tag{33}$$

$$F_{\text{end}} = 0 \tag{34}$$

Equation (29) could be re-written as

$$\begin{Bmatrix} x_{\text{end}} \\ 0 \\ 1 \end{Bmatrix} = \left[\begin{array}{cc|c} \mathbf{U}_{\text{sys11}} & \mathbf{U}_{\text{sys12}} & \mathbf{U}_{\text{sys13}} \\ \mathbf{U}_{\text{sys21}} & \mathbf{U}_{\text{sys22}} & \mathbf{U}_{\text{sys23}} \\ \hline \mathbf{U}_{\text{sys31}} & \mathbf{U}_{\text{sys32}} & \mathbf{U}_{\text{sys33}} \end{array}\right] \begin{Bmatrix} 0 \\ F_{\text{base}} \\ 1 \end{Bmatrix} \tag{35}$$

F_{base} can be found from row 2 of this matrix equation:

$$F_{\text{base}} = \frac{-\mathbf{U}_{\text{sys23}}}{\mathbf{U}_{\text{sys22}}} \tag{36}$$

Once F_{base} is known, the state vector at the base is also known

$$\mathbf{z}_{\text{base}} = \begin{Bmatrix} 0 \\ F_{\text{base}} \\ 1 \end{Bmatrix} \tag{37}$$

Once $\mathbf{z}_{\text{base}}$ is known, that state vector at any location in the system can be found. For example,

$$\mathbf{z}_{\text{m1}} = \mathbf{U}_{\text{m1}}\, \mathbf{U}_{\text{sd1}}\, \mathbf{z}_{\text{base}} \tag{38}$$

and

$$\mathbf{z}_{\text{m2}} = \mathbf{z}_{\text{end}} = \mathbf{U}_{\text{sys}}\, \mathbf{z}_{\text{base}} \tag{39}$$

where $\mathbf{z}_{\text{m1}}$ and $\mathbf{z}_{\text{m2}}$ are the state vectors immediately to the right of masses 1 and 2, respectively.

The transfer functions x_1/F and x_2/F can be found be dividing the first element of the state vectors by F:

$$\frac{x_1}{F} = \frac{\mathbf{z}_{\text{m1}}[1]}{F} = \frac{m_2\, s^2 + c_2\, s + k_2}{D} \tag{40}$$

and

$$\frac{x_2}{F} = \frac{\mathbf{z}_{\text{end}}[1]}{F} = \frac{c_2\, F\, s + F\, k_2}{D} \tag{41}$$

where

$$\begin{aligned} D &= m_1\, m_2\, s^4 + [(c_2 + c_1)\; m_2 + c_2\, m_1]\; s^3 + \\ &\quad [(k_2 + k_1)\; m_2 + k_2\, m_1 + c_1\, c_2]\; s^2 + (c_1\, k_2 + c_2\, k_1)\; s + k_1\, k_2 \end{aligned} \tag{42}$$

1.3.4 Classical Analysis

These TMM analysis results can be verified by classical transfer function analysis. The equations of motion of the system are

$$m_1\, s^2\, x_1 = c_2\, s\; (x_2 - x_1) + k_2\; (x_2 - x_1) - c_1\, s\, x_1 - k_1\, x_1 + F \tag{43}$$

$$m_2\, s^2\, x_2 = -c_2\, s\; (x_2 - x_1) - k_2\; (x_2 - x_1) \tag{44}$$

Solving equations 43 and 44 for x_1 and x_2 and dividing by F gives

$$\frac{x1}{F} = \frac{m_2\, s^2 + c_2\, s + k_2}{D} \tag{45}$$

$$\frac{x2}{F} = \frac{c_2\, s + k_2}{D} \tag{46}$$

where

$$\begin{aligned} D &= m_1\, m_2\, s^4 + [(c_2 + c_1)\; m_2 + c_2\, m_1]\; s^3 + \\ &\quad [(k_2 + k_1)\; m_2 + k_2\, m_1 + c_1\, c_2]\; s^2 + (c_1\, k_2 + c_2\, k_1)\; s + k_1\, k_2 \end{aligned} \tag{47}$$

1.3.5 More Complicated Systems

The classical analysis is clearly shorter and more straightforward for this simple system. This simple system was chosen to illustrate the procedure for forced-response analysis using the TMM. The procedure does not change for systems with more states or more elements.

For systems with N states, equation (35) will have $N+1$ rows (because of the augmented transfer matrices), and $N/2$ of those rows will have 0's on the left hand side of the equation. These $N/2$ equations can be solved for the unknown portions of the state vector at the base. Once the state vector at the base is known, the state vector at any location in the system can be found.

CHAPTER II

HYDRAULIC ACTUATOR MODELING

2.1 Introduction

Actuator dynamics and actuator/structure interaction are a significant part of the challenge of modeling flexible robots. The actuators must be modeled accurately, but in a way that is compatible with the overall system model. Modeling an actuator with its own transfer matrix will allow seamless integration into a TMM system model. Transfer functions for the actuator can be converted to transfer matrices fairly easily, provided that the transfer functions are in terms of the state variables already being used in the system transfer matrix model (i.e. lateral displacement w, rotation θ, bending moment M, and shear force V). A transfer function in terms of these variables would be an ideal model from the standpoint of integration with the TMM.

2.2 Initial Model

SAMII's hydraulic actuators are rotary motors controlled by servo-valves. A schematic of an actuator is shown in Figure 6. The input is a voltage v that is proportional to spool position d. The output is angle of rotation θ. This work focuses on modeling and control design for one of SAMII's hydraulic actuators. The initial model for an actuator was a velocity source with a first-order lag. The transfer function for this model takes the form:

$$\frac{\theta}{v} = \frac{Kp}{s(s+p)} \tag{48}$$

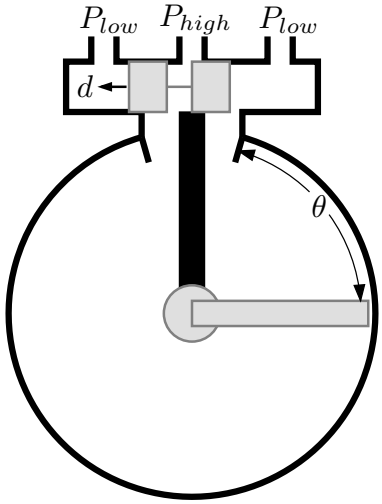

Figure 6: One of SAMII's hydraulic actuators, a rotary hydraulic motor controlled by a servo-valve

This model can easily be made into a transfer matrix model with an augmented input:

$$\mathbf{U} = \left[\begin{array}{cccc|c} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \frac{K\,p\,v}{s(s+p)} \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 \end{array}\right] \tag{49}$$

An augmented transfer matrix has an additional column used to model external forcing. For an element without external forcing, the column is all zeros except for the bottom element, which is one. An additional row is also added to keep the matrices square and allow multiplication of transfer matrices. The state vector for this model has an additional one added to it as well:

$$\mathbf{z} = \left\{\begin{array}{c} w \\ \theta \\ M \\ V \\ 1 \end{array}\right\} \tag{50}$$

The states before and after this actuator are related by

$$\mathbf{z}_{\text{after}} = \mathbf{U}\,\mathbf{z}_{\text{before}} \tag{51}$$

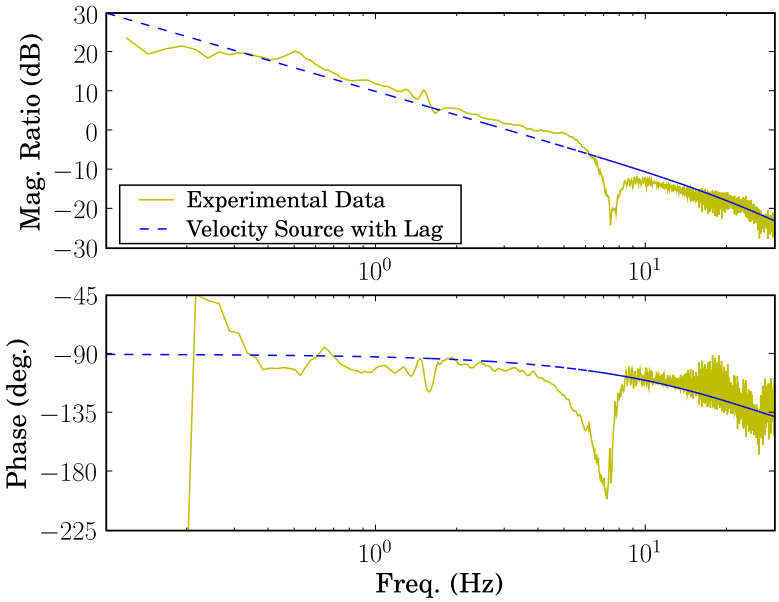

Figure 7: Comparison of actuator Bode plots θ/v from experimental data and from a model were the actuator is an angular velocity source with a first-order lag.

The second row of this matrix equation gives

$$\theta_{\text{after}} = \theta_{\text{before}} + \frac{K\,p\,v}{s(s+p)} \tag{52}$$

or

$$\theta = \frac{K\,p\,v}{s(s+p)} \tag{53}$$

where $\theta = \theta_{\text{after}} - \theta_{\text{before}}$ is the relative angle across the actuator.

Figure 7 shows that this model is accurate over much of the frequency range, but it breaks down near the second structural natural frequency at approximately 7 Hz. There is a 90° difference between the model and actual phase lag for the actuator near this frequency. This is a significant phase error that could affect the ability of the model to predict system stability.

Note that for all experimental Bode plots in this work, the system is actuated by a swept sine input to the desired angle for joint 2. All other joints are under feedback control, but the desired position is held constant. The control of the other joints holds SAMII in a nominal position. A picture of SAMII highlighting joint 2 is shown in Figure 8. The swept sine amplitude for joint 2 was typically 1.5°. Excitation amplitudes from 0.5–3.0° were

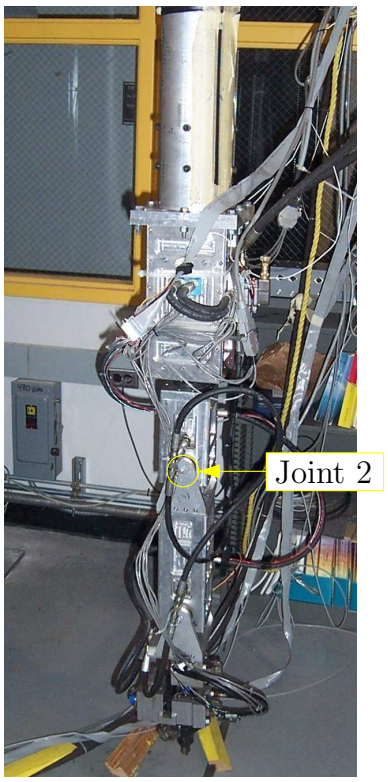

Figure 8: Picture of SAMII showing joint 2, the actuated joint for the experimental Bode plots in this thesis.

tried. Small excitation amplitudes led to significantly different results when the control system did not compensate for the dead-zone of the actuator. All testing was done at room temperature and with nominal hydraulic pressure of 1500 psi.

2.3 An Actuator Model with Intrinsic Velocity Feedback

One potential problem with the velocity source model is that it assumes that the actuator can supply any magnitude of torque required. A model assuming intrinsic velocity feedback has been proposed to improve the fidelity of the actuator model:

$$T = g_v \left(g_a \, v - \dot{\theta} \right) \tag{54}$$

where T is the torque applied by the actuator, v is the voltage to the actuator, $\dot{\theta}$ is the angular velocity of the joint, and g_v and g_a are constants to be determined. A block diagram of this model is shown in Figure 9.

If $g_v \rightarrow \infty$, this model is equivalent to the velocity source model. For finite g_v, the

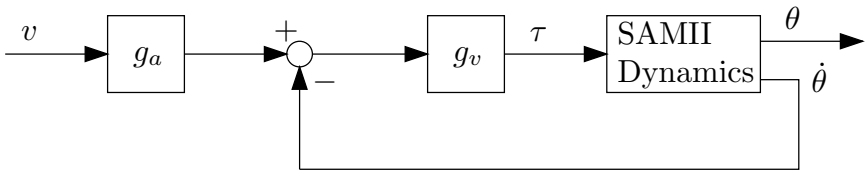

Figure 9: Block diagram of a hydraulic actuator model that assumes intrinsic velocity feedback.

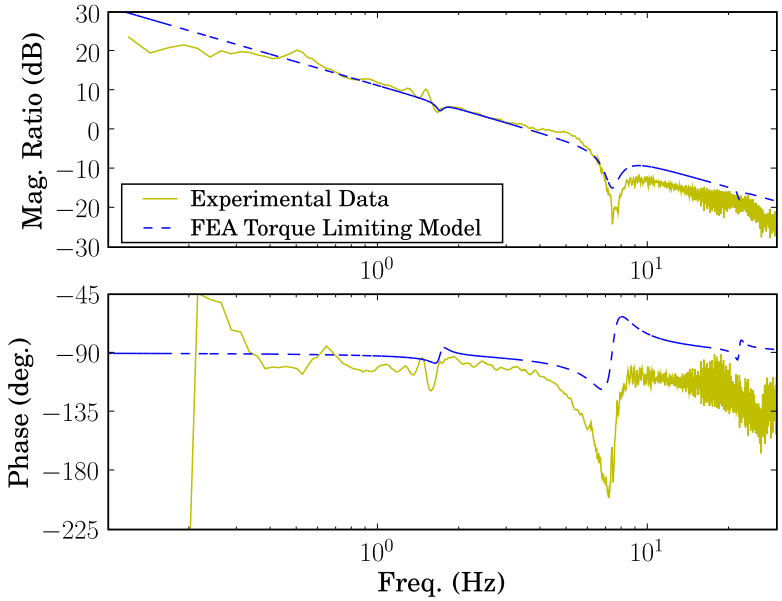

Figure 10: Comparison of actuator Bode plots θ/v from experimental data and from a torque limiting FEA model.

transfer function between v and θ is affected by the plant dynamics near the resonances.

Figure 10 shows the results of using equation (54) in an FEA model of the system. This model shows some improvement over the velocity source model of Figure 7: there is interaction between the actuator and structure at resonances. However, there is still significant error in the phase of the model. The model predicts a phase dip at the second natural frequency of 20–30° followed by an increase in phase above the nominal value before resonance of the same magnitude. The experimental results show a phase dip of about 90° at the second natural frequency, followed by a phase recovery but no increase.

This same approach of modeling the actuator with intrinsic velocity feedback was taken by Obergfell [47]. Figure 11 compares modeled and experimental Bode plots from [47] for

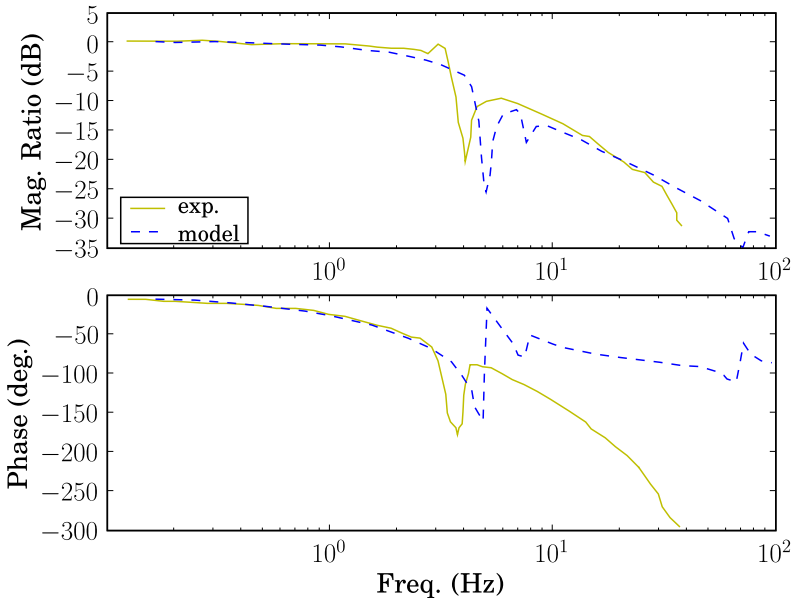

Figure 11: Comparison of experimental and model Bode plots for an intrinsic velocity feedback model of a hydraulic actuator by Obergfell [47].

the hydraulic actuator of RALF. While the dynamics of RALF are quite different from those of SAMII, the intrinsic velocity feedback model also predicts a phase dip followed by an increase in phase for RALF as it did for SAMII. This phase increase after resonance does not agree with experiment for either robot.

Attempting to better model the actuator torque dynamics seems to show promise, but the model of equation (54) does not seem to be the final answer. An experimental investigation was undertaken to better understand the torque dynamics of the actuator.

2.4 Torque Experiments

Figure 12 shows SAMII in the configuration used for earlier investigations and with all of links 0–6 attached. Figure 13 shows the setup for the torque experiment testing. SAMII's links 2-6 have been removed and replace with one, small, rigid link. There is a force/torque sensor installed between this rigid link and the output shaft of joint 2. Note that SAMII's base was rigidly connected to the floor for this testing (unless explicitly indicated otherwise). The torque testing was done in a vertical and a horizonal configuration. Figure 14 shows the difference between these two.

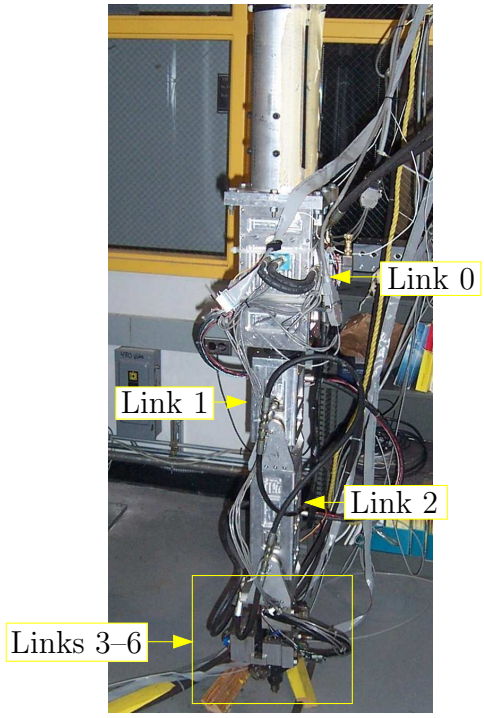

Figure 12: SAMII in a configuration used for earlier investigations.

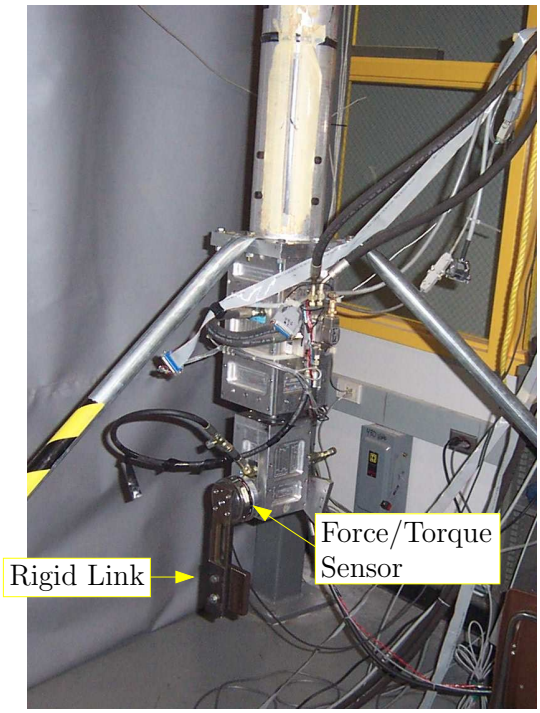

Figure 13: Set-up for testing in a vertical configuration.

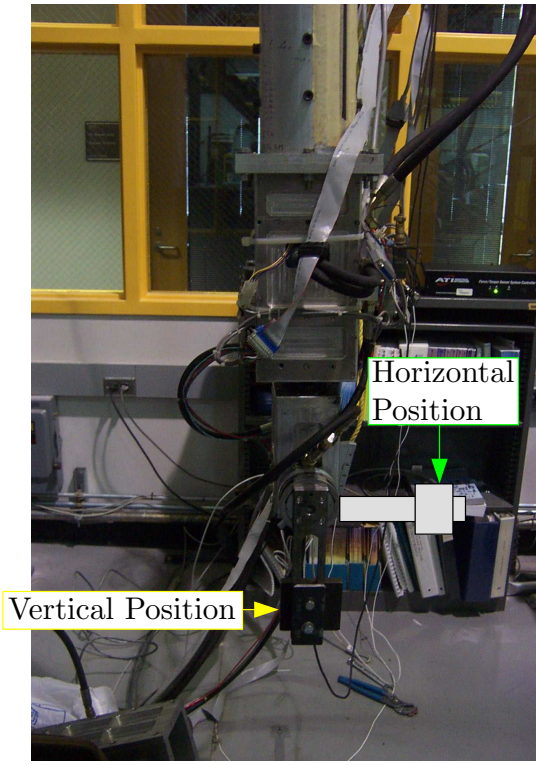

Figure 14: Set-up for testing in a vertical configuration (as pictured) and a horizonal configuration (illustrated).

The vertical configuration was chosen first because it resembles the configuration used in previous testing. As seen in Figures 15 and 16, the vertical configuration leads to somewhat unexpected results. Considerable effort has been made to remove flexibility from the system, but there is a very pronouced spring effect. The horizontal configuration was tried in an attempt to explain where the spring effect was coming from. In the vertical configuration, gravity contributes a small torque, but one that is proportional to θ and is a restoring torque. In the horizontal configuration, the gravity torque is larger, but nearly constant.

Figure 15 shows the Bode plots for θ/M_z for the horizonal and vertical configurations. These Bode plots show the relationship between torque generated by the actuator and the resulting angular displacement. There is not an obvious input/output relationship between these two variables across the two different configurations.

Figure 16 shows the Bode plot for torque M_z vs. input voltage to the actuator v. For

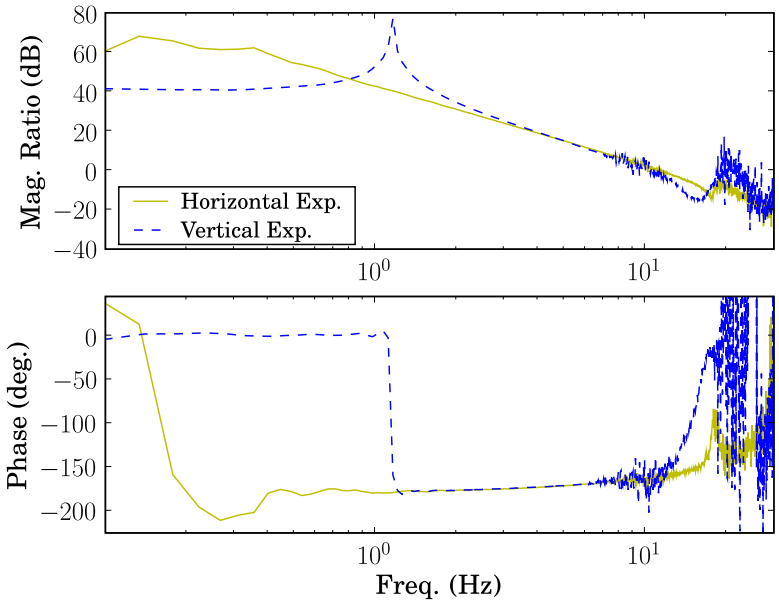

Figure 15: Bode plot of θ/M_z for the hydraulic actuator in horizonal and vertical configurations.

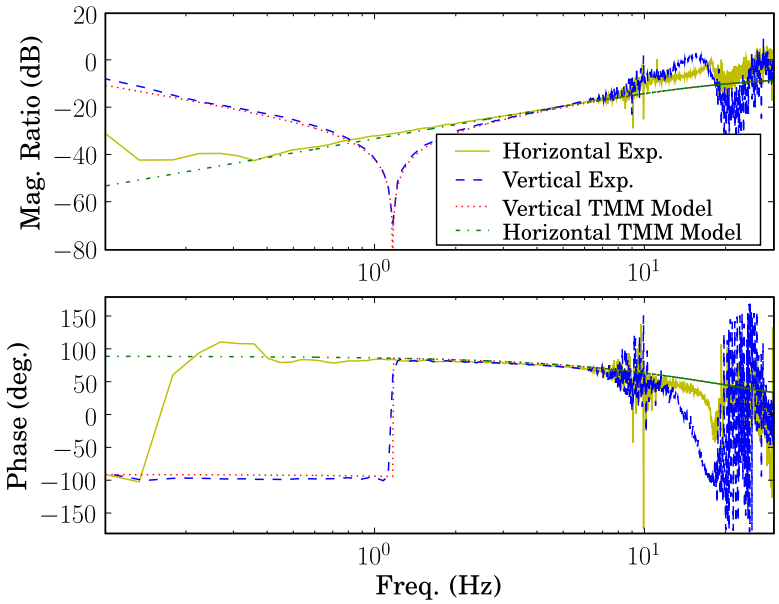

Figure 16: Bode plot of M_z/v for the hydraulic actuator in horizonal and vertical configurations.

the vertical configuration, the sum of moments about the joint gives

$$J\ddot{\theta} + mgr\theta = M_z \tag{55}$$

where m is the mass of the rigid link, g is the acceleration due to gravity, r is the distance from the joint to the center of gravity of the link, and J is the second moment of inertia of the link.

At low frequencies, the gravity part of this equation dominants. This leads to torque that is proportional to θ. Since v is proportional to $\dot{\theta}$, $M_z/v \approx 1/s$. At high frequencies, the inertia torque dominates and M_z is proportional to $\ddot{\theta}$. This leads to $M_z/v \approx s$. These approximate low and high frequency relationships are seen in Figure 16 and the TMM model is seen to agree closely with experiment for both the vertical and horizonal positions.

Figure 17 shows the Bode plots for angular position θ vs. input voltage to the actuator v. In both the horizontal and vertical configurations, the actuator acts as an integrator or a velocity source with $\dot{\theta}$ proportional to v. This is the actuator relationship that is most apparent and does not change from horizontal to vertical configurations. Taking Figures 16 and 17 together suggests that once the relationship between θ and v is established by the actuator, the actuator simply generates whatever torque is necessary to bring about the motion. The angular velocity source model seems fundamentally sound and allowing the model torque to simply be whatever results from this motion leads to good agreement between model and experiment.

It seems that in order to explain the interaction between the actuator and structure at resonance, there is some compliance that needs to be added to the actuator model to allow for some back-driving of the actuator. To that end, the next section shows the results of combining an angular velocity source with a torsional spring/damper.

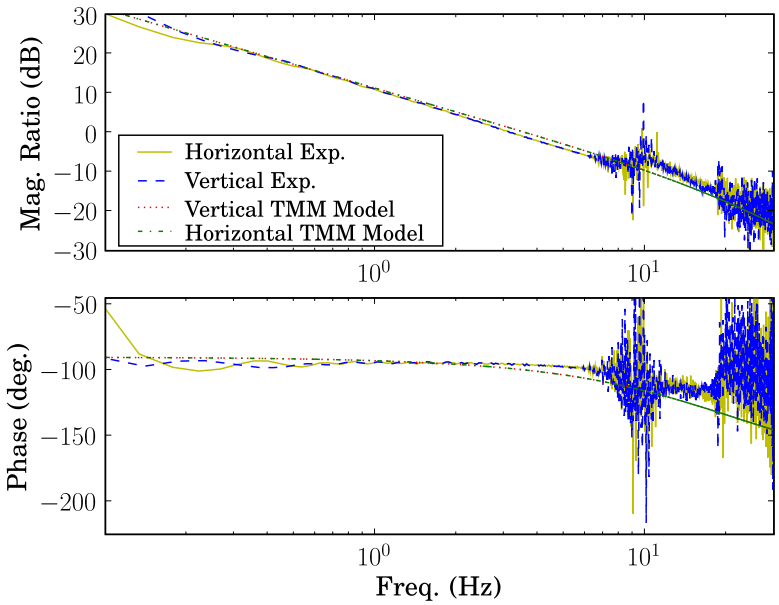

Figure 17: Bode plot of θ/v for the hydraulic actuator in horizonal and vertical configurations.

2.5 Angular Velocity Source in Series with a Torsional Spring/-Damper

Modeling the hydraulic actuator as an angular velocity source in series with a torsional spring damper would produce a transfer function of the form

$$\theta = \frac{K\,p\,v}{s(s+p)} + \frac{M}{cs+k} \tag{56}$$

This could be put in transfer matrix form:

$$\mathbf{U}_{\text{ol}} = \left[\begin{array}{cccc|c} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & \dfrac{1}{cs+k} & 0 & \dfrac{K\,p\,v}{s(s+p)} \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 \end{array}\right] \tag{57}$$

The parameters for this model are determined by a system identification procedure that is discussed in depth in chapter 7. Figure 18 compares a Bode plot from this model with one from experimental data. This model has been inferred from experimental results where

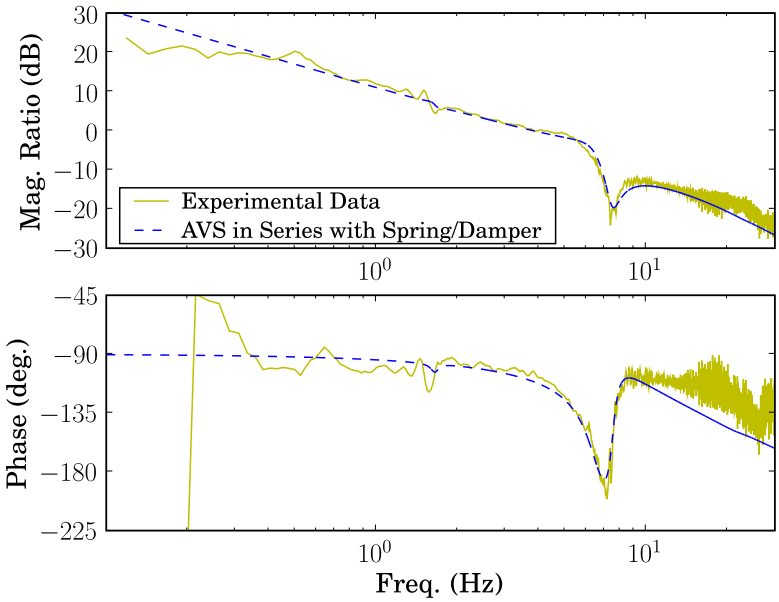

Figure 18: Comparison of actuator Bode plots θ/v from experimental data and from a transfer matrix model where the actuator is an angular velocity source in series with a spring/damper element.

the angular velocity source model captures much of the dynamics of the system, but fails to capture actuator compliance that seems apparent at resonance. This model leads to close agreement between model and experiment.

Figure 19 compares the angular velocity source in series with the spring/damper to the FEA model mentioned previously. This latest model seems to better capture the dynamics at resonance.

While the model of equation (57) is a fairly simple one, it accurately captures the actuator dynamics and the interaction between the actuator and the structure. Figure 18 experimentally verifies this model. Furthermore, the TMM provides a convenient way to combine the effects of the actuator dynamics with the dynamics of flexible links and the rest of the structure.

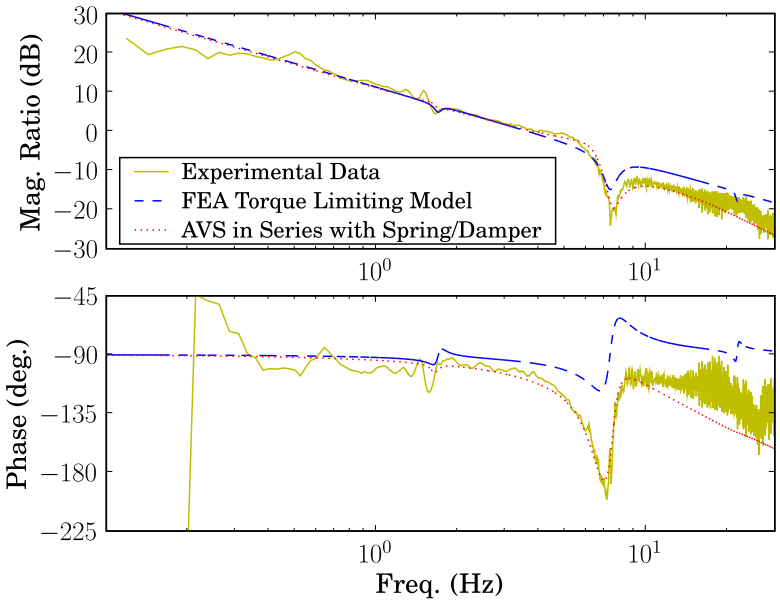

Figure 19: Overlay of the results from Figures 10 and 18, showing that the angular velocity source in series with a spring/damper element appears to better capture the dynamics than the torque limiting FEA model.

CHAPTER III

NON-COLLOCATED FEEDBACK MODELING

3.1 Introduction

This chapter explains how the transfer matrix method was expanded to correctly model practical control schemes for flexible robots involving non-collocated feedback. Non-collocated feedback refers to a case where the sensor and actuator are not at exactly the same physical location. Non-collocated feedback is risky from the standpoint of system stability. However, there are often practical reasons why sensors and actuators cannot be precisely at the same location.

The control of SAMII involves two cases where it is not practical to precisely collocate actuators and sensors: angular position feedback for a hydraulic actuator with internal compliance and acceleration feedback for vibration suppression. In the case of the hydraulic actuator, non-collocated feedback is practically unavoidable because the compliance is internal to the actuator. In the vibration suppression case, all of the joints could be used in combination to develop a controller that suppresses vibration of the base. Mounting an accelerometer at each joint, in order to collocate the sensor and actuator, would be costly and would pose significant wire-routing challenges. It would also consume significant computational and data acquisition resources to handle so many signals (possibly three accelerometers per joint). As a result, non-collocated feedback is necessary for practical reasons in SAMII's case and it poses an interesting challenge for the TMM that should be overcome for the sake of applying the method to a broader range of problems.

Figure 20 shows a picture of SAMII labeling joint 2 and the two outputs of concern for this section. The input to the system is the command voltage v to joint 2's hydraulic actuator. θ is the angular displacement of joint 2 and $\ddot{x}$ is the acceleration of SAMII's base (i.e. the end of the cantilever beam). This acceleration is fed back into the controller for joint 2 in the vibration suppression scheme. The entire control scheme, including vibration

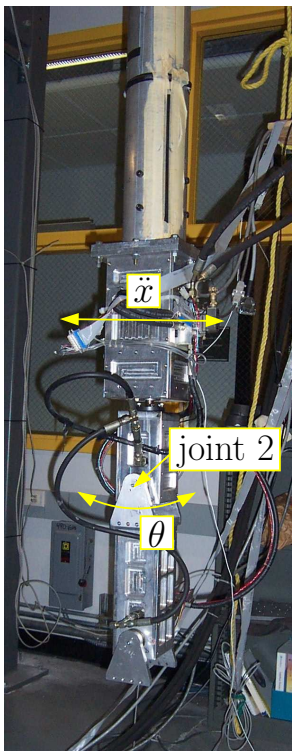

Figure 20: Picture of SAMII showing joint 2 and the two outputs: θ is the angular position of joint 2 and $\ddot{x}$ is the acceleration of SAMII's base (i.e. the end of the cantilever beam).

suppression is shown in the block diagram of Figure 21. θ and $\ddot{x}$ are fed back in two separate control loops. This controller uses one degree of freedom of SAMII to damp vibration in one direction. It is very similar to the control schemes used in [18], [37], and [23].

The approach for modeling the θ feedback control portion of SAMII using the transfer matrix method will be derived in the next section. This derivation will be experimentally verified in section 3.4. The approach for modeling non-collocated $\ddot{x}$ feedback will be discussed in section 3.3 with experimental verification in section 3.4.3.

3.2 Position Control - without Vibration Suppression

The open-loop response of a hydraulic actuator with flexibility is given by

$$\theta_{\text{after}} - \theta_{\text{before}} = \theta = G_{\text{act}} v + \frac{M}{cs + k} \tag{58}$$

where v is the input voltage to the hydraulic actuator. θ refers to the actual relative displacement of the joint. This is the same as θ in Figure 20. c and k are the damping and

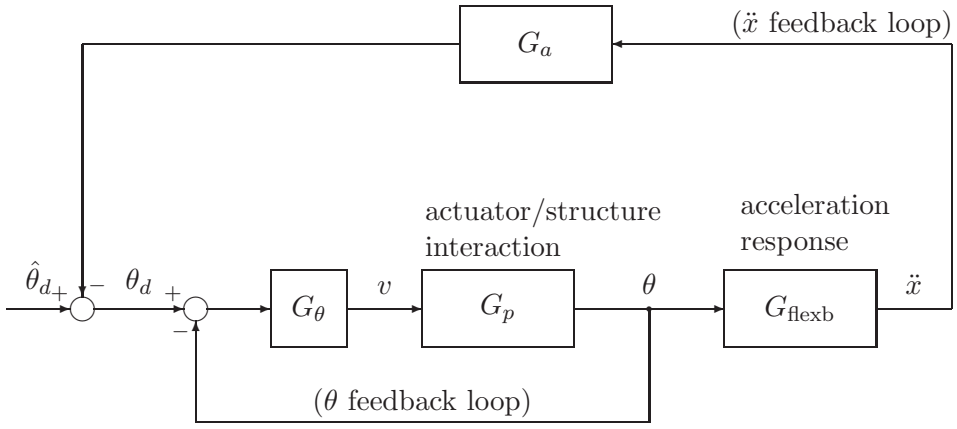

Figure 21: Block diagram of the system with position control (θ feedback) and vibration suppression ($\ddot{x}$ feedback).

stiffness coefficients that model compliance in the actuator. M is the moment across the actuator and G_{act} is the transfer function θ/v for the actuator. Note that G_{act} refers to the transfer function of the actuator in isolation, while G_p in the block diagrams of Figures 21, 23, and 28 models the actuator interacting with the rest of the structural model.

An augmented open-loop transfer matrix would take the form

$$\mathbf{U}_{\text{ol}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & \dfrac{1}{cs+k} & 0 & G_{\text{act}}v \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{59}$$

where the states are

$$\mathbf{z} = \begin{Bmatrix} x \\ \theta \\ M \\ V \\ 1 \end{Bmatrix} \tag{60}$$

Relative position feedback control (θ feedback) could be accomplished by setting

$$v = G_\theta \left(\theta_d - \theta\right) \tag{61}$$

where θ_d refers to the desired relative joint displacement. Substituting this expression for v into equation (58) and solving for θ gives

$$\theta = \frac{G_\theta G_{\text{act}} \theta_d}{G_\theta G_{\text{act}} + 1} + \frac{M}{(G_\theta G_{\text{act}} + 1)(cs + k)} \tag{62}$$

and the closed-loop transfer matrix for θ feedback would be

$$\mathbf{U}_{\text{cl}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & \dfrac{1}{(G_\theta G_{\text{act}} + 1)(cs + k)} & 0 & \dfrac{G_\theta G_{\text{act}}\, \theta_d}{G_\theta G_{\text{act}} + 1} \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{63}$$

This result can be generalized for the open-loop case

$$\mathbf{U}_{\text{ol}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ \alpha & 1 & \beta & \gamma & G_{\text{act}} v \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{64}$$

where α, β, and γ represent the dynamics between θ and linear displacement x, bending moment M, and shear force V:

$$\theta_{\text{after}} = V\gamma + \alpha x + G_{\text{act}} v + \theta_{\text{before}} + \beta M \tag{65}$$

Using v from equation (61) and again solving for θ gives

$$\theta = \frac{V\gamma + \alpha x + G_\theta G_{\text{act}} \theta_d + \beta M}{G_\theta G_{\text{act}} + 1} \tag{66}$$

so that the general closed-loop transfer matrix is

$$\mathbf{U}_{\text{cl}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ \dfrac{\alpha}{G_\theta G_{\text{act}} + 1} & 1 & \dfrac{\beta}{G_\theta G_{\text{act}} + 1} & \dfrac{\gamma}{G_\theta G_{\text{act}} + 1} & \dfrac{G_\theta G_{\text{act}}}{G_\theta G_{\text{act}} + 1} \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{67}$$

For the specific case of relative θ feedback with torsional compliance

$$\alpha = 0 \tag{68}$$

$$\beta = \frac{1}{cs+k} \tag{69}$$

$$\gamma = 0 \tag{70}$$

and

$$\mathbf{U}_{\mathrm{cl}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & \dfrac{1}{(cG_\theta G_{\mathrm{act}} + c)\, s + (G_\theta G_{\mathrm{act}} + 1)\, k} & 0 & \dfrac{G_\theta G_{\mathrm{act}}}{G_\theta G_{\mathrm{act}} + 1} \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{71}$$

and for the even more specific case of a hydraulic actuator the actuator transfer function G_{act} would likely take the form

$$G_{\mathrm{act}} = \frac{Kp}{s(s+p)} \tag{72}$$

3.3 With Vibration Suppression

Vibration suppression will be based on accelerometer feedback as shown in the block diagram of Figure 21. Accelerometer feedback will be added to the desired joint motion according to

$$\theta_d = G_a s^2 x_{\mathrm{beam}} + \hat{\theta}_d \tag{73}$$

Substituting this expression for θ into equation (61) produces

$$v = G_\theta \left(G_a s^2 x_{\mathrm{beam}} - \theta + \hat{\theta}_d \right) \tag{74}$$

Substituting this value for v into equation (58) gives

$$\theta = G_\theta G_{\mathrm{act}} \left(G_a s^2 x_{\mathrm{beam}} - \theta + \hat{\theta}_d \right) + \frac{M}{cs+k} \tag{75}$$

Solving for θ,

$$\theta = \frac{G_a G_\theta G_{\mathrm{act}} s^2 x_{\mathrm{beam}}}{G_\theta G_{\mathrm{act}} + 1} + \frac{G_\theta G_{\mathrm{act}} \hat{\theta}_d}{G_\theta G_{\mathrm{act}} + 1} + \frac{M}{(cG_\theta G_{\mathrm{act}} + c)\, s + (G_\theta G_{\mathrm{act}} + 1)\, k} \tag{76}$$

This control scheme can be modeled as the product of two transfer matrices, $\mathbf{U}_{\text{cl}}$ from equation (63) and

$$\mathbf{U}_{\text{acc}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & \dfrac{G_a G_\theta G_{\text{act}} s^2 x_{\text{beam}}}{G_\theta G_{\text{act}} + 1} \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{77}$$

Multiplying these matrices in either order

$$\mathbf{z}_{\text{after}} = \mathbf{U}_{\text{acc}} \mathbf{U}_{\text{cl}} \mathbf{z}_{\text{before}} \tag{78}$$

or

$$\mathbf{z}_{\text{after}} = \mathbf{U}_{\text{cl}} \mathbf{U}_{\text{acc}} \mathbf{z}_{\text{before}} \tag{79}$$

gives

$$\mathbf{z}_{\text{after}} = \begin{Bmatrix} x_{\text{before}} \\ \dfrac{G_a G_\theta G_{\text{act}} s^2 x_{\text{beam}}}{G_\theta G_{\text{act}} + 1} + \theta_{\text{before}} + \dfrac{M_{\text{before}}}{(G_\theta G_{\text{act}} + 1)(cs + k)} + \dfrac{G_\theta G_{\text{act}}}{G_\theta G_{\text{act}} + 1} \\ M_{\text{before}} \\ V_{\text{before}} \\ 1 \end{Bmatrix} \tag{80}$$

The only problem is that $\mathbf{U}_{\text{acc}}$ depends on x_{beam} which is not a state available at this location in the TMM model. x_{beam} needs to be expressed in terms of the state variables available at the location immediately preceding $\mathbf{U}_{\text{acc}}$.

Figure 22 shows a schematic of the system with the elements of the transfer matrix model labeled. Each element on the schematic has a corresponding transfer matrix that transfers the state vector from one end of the element to the other. $\mathbf{z}_{\text{base}}$ is the state vector at the base of the system. The state vector after the base spring would be

$$\mathbf{z}_{\text{basespring}} = \mathbf{U}_{\text{basespring}}\, \mathbf{z}_{\text{base}} \tag{81}$$

$\mathbf{z}_{\text{basespring}}$ is the state vector in between the base spring and the beam. The state vector at the end of the beam is then

$$\mathbf{z}_{\text{beam}} = \mathbf{U}_{\text{beam}}\, \mathbf{z}_{\text{basespring}} \tag{82}$$

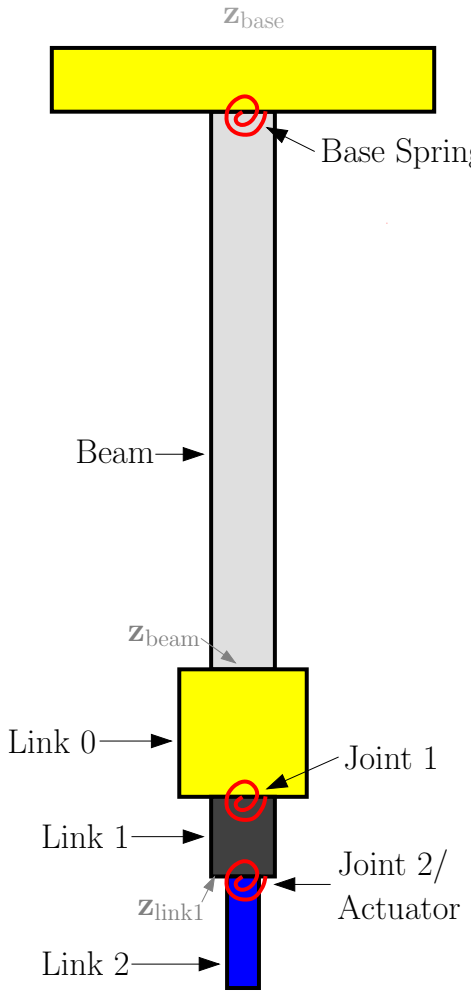

Figure 22: Schematic of SAMII showing showing transfer matrix elements.

or

$$\mathbf{z}_{\text{beam}} = \mathbf{U}_{\text{beam}}\,\mathbf{U}_{\text{basespring}}\,\mathbf{z}_{\text{base}} \tag{83}$$

The overall system model, including position feedback and vibration suppression, will take the form

$$\mathbf{z}_{\text{tip}} = \mathbf{U}_{\text{link2}}\,\mathbf{U}_{\text{cl}}\,\mathbf{U}_{\text{acc}}\,\mathbf{U}_{\text{link1}}\,\mathbf{U}_{\text{joint1}}\,\mathbf{U}_{\text{link0}}\,\mathbf{U}_{\text{beam}}\,\mathbf{U}_{\text{basespring}}\,\mathbf{z}_{\text{base}} \tag{84}$$

The state vector available to multiply the acceleration feedback matrix $\mathbf{U}_{\text{acc}}$ is $\mathbf{z}_{\text{link1}}$ as shown in Figure 22. $\mathbf{z}_{\text{link1}}$ can be expressed in terms of the states at the end of the beam as

$$\mathbf{z}_{\text{link1}} = \mathbf{U}_{\text{link1}}\,\mathbf{U}_{\text{joint1}}\,\mathbf{U}_{\text{link0}}\,\mathbf{z}_{\text{beam}} \tag{85}$$

$\mathbf{z}_{\text{beam}}$ can then be written in terms of $\mathbf{z}_{\text{link1}}$ as

$$\mathbf{z}_{\text{beam}} = \left(\mathbf{U}_{\text{link1}}\,\mathbf{U}_{\text{joint1}}\,\mathbf{U}_{\text{link0}}\right)^{-1}\,\mathbf{z}_{\text{link1}} \tag{86}$$

assuming the transfer matrices $\mathbf{U}_{\text{link1}}$, $\mathbf{U}_{\text{joint1}}$,and $\mathbf{U}_{\text{link0}}$ are known. Links 0 and 1 are rigid masses and joint 1 is an unactuated joint (for the purposes of this analysis). A rigid mass transfer matrix is

$$\mathbf{U}_{\text{R}} = \begin{bmatrix} 1 & L & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -M\left(L-r\right)s^2 & Izs^2 - M\left(L-r\right)rs^2 & 1 & -L & 0 \\ Ms^2 & Mrs^2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{87}$$

and for an unactuated joint

$$\mathbf{U}_{\text{J}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & \dfrac{1}{cs+k} & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{88}$$

so that for the needs of equation (86),

$$\mathbf{U}_{\text{link0}} = \begin{bmatrix} 1 & L_0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -m_0\left(L_0-r_0\right)s^2 & Iz_0s^2 - m_0\left(L_0-r_0\right)r_0s^2 & 1 & -L_0 & 0 \\ m_0s^2 & m_0r_0s^2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{89}$$

$$\mathbf{U}_{\text{joint1}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & \dfrac{1}{c_1s+k_1} & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{90}$$

$$\mathbf{U}_{\text{link1}} = \begin{bmatrix} 1 & L_1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -m_1\left(L_1 - r_1\right)s^2 & Iz_1 s^2 - m_1\left(L_1 - r_1\right)r_1 s^2 & 1 & -L_1 & 0 \\ m_1 s^2 & m_1 r_1 s^2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{91}$$

Multiplying by the state vector as shown in equation (86) gives

$$\begin{Bmatrix} x_{\text{beam}} \\ \theta_{\text{beam}} \\ M_{\text{beam}} \\ V_{\text{beam}} \\ 1 \end{Bmatrix} = \left(\mathbf{U}_{\text{link1}}\,\mathbf{U}_{\text{joint1}}\,\mathbf{U}_{\text{link0}}\right)^{-1} \begin{Bmatrix} x_{l1} \\ \theta_{l1} \\ M_{l1} \\ V_{l1} \\ 1 \end{Bmatrix} \tag{92}$$

This control scheme needs x_{beam}:

$$x_{\text{beam}} = \frac{N_{\text{xb}}}{D_{\text{xb}}} \tag{93}$$

where

$$\begin{aligned} N_{x_b} \;=\; & \left(-L_0 m_1 r_1 s^2 + c_1 s + k_1\right) x_{l1} + L_0 L_1 V_{l1} + \\ & \left[\left(-L_0 m_1 r_1^2 + L_0 L_1 m_1 r_1 - Iz_1 L_0\right) s^2 - c_1\left(L_1 + L_0\right) s - k_1(L_1 + L_0)\right] \theta_{l1} \\ & + L_0 M_{l1} \end{aligned} \tag{94}$$

and

$$D_{x_b} = c_1 s + k_1 \tag{95}$$

Note that while N_{xb} appears to be of a higher order than D_{xb}, the terms in N_{xb} are multiplying the states x_{l1}, θ_{l1}, V_{l1}, and M_{l1} whose denominators will be of higher order than their numerators.

Defining

$$a = -\frac{L_0 m_1 r_1 s^2 - c_1 s - k_1}{c_1 s + k_1} \tag{96}$$

$$b = -\frac{\left(L_0 m_1 r_1^2 - L_0 L_1 m_1 r_1 + Iz_1 L_0\right) s^2 + c_1\left(L_1 + L_0\right) s + k_1(L_1 + L_0)}{c_1 s + k_1} \tag{97}$$

$$c = \frac{L_0}{c_1 s + k_1} \tag{98}$$

$$d = \frac{L_0 L_1}{c_1 s + k_1} \tag{99}$$

allows $\mathbf{U}_{\text{acc}}$ from equation (77) to be written in terms of the states of link 1 as

$$\mathbf{U}_{\text{acc}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ \frac{a\,G_a G_\theta G_{\text{act}} s^2}{G_\theta G_{\text{act}} + 1} & \frac{b\,G_a G_\theta G_{\text{act}} s^2}{G_\theta G_{\text{act}} + 1} + 1 & \frac{c\,G_a G_\theta G_{\text{act}} s^2}{G_\theta G_{\text{act}} + 1} & \frac{d\,G_a G_\theta G_{\text{act}} s^2}{G_\theta G_{\text{act}} + 1} & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{100}$$

where

$$x_{\text{beam}} = a x_{l1} + b\theta_{l1} + c M_{l1} + d V_{l1} \tag{101}$$

Equation (100) now represents a legitimate transfer matrix model of acceleration feedback that can be used in the system model of equation (84).

3.4 Experimental Verification

The TMM models derived in the previous section will now be verified experimentally and with transfer function analysis.

3.4.1 System ID

The first step in experimental verification of the closed-loop system models is to determine unknown parameters based on system identification (system ID is discussed extensively in chapter 7). Open-loop Bode responses are used for this purpose. The system parameters are then used in models that predict the closed-loop Bode responses as the two successive control loops are closed: the position control loop based on θ feedback and the vibration suppression loop based on acceleration feedback.

The block diagram for the open-loop system is shown in Figure 23. The open-loop actuator Bode plot (Figure 24) represents G_p while the Bode plot in Figure 25 represents $G_p G_{\text{flexb}}$. G_p includes the interaction between the actuator and the structure.

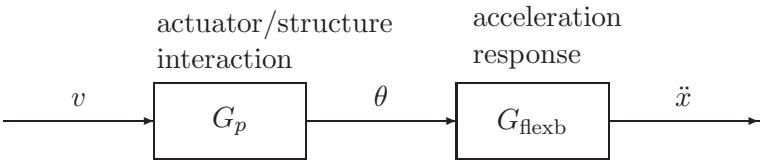

Figure 23: Block diagram of the open-loop system.

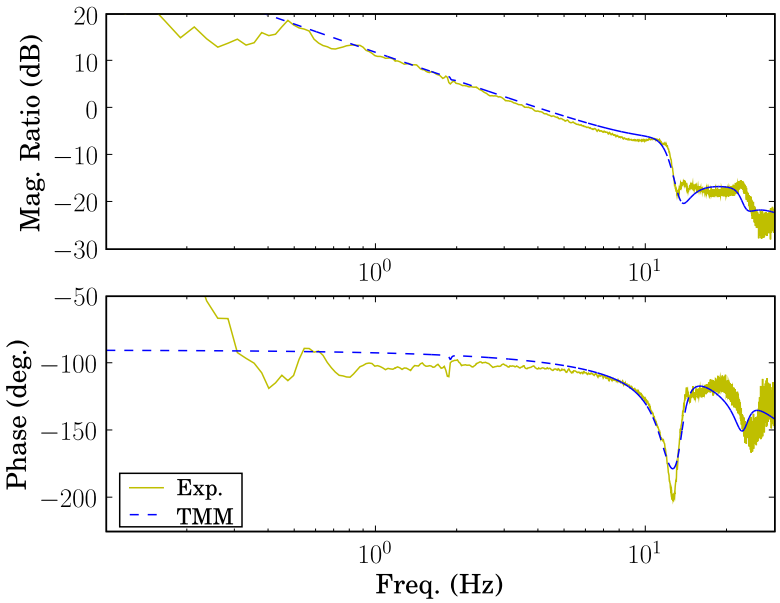

Figure 24: Open-loop actuator Bode plot θ/v comparing experimental data to the TMM model after curve fitting.

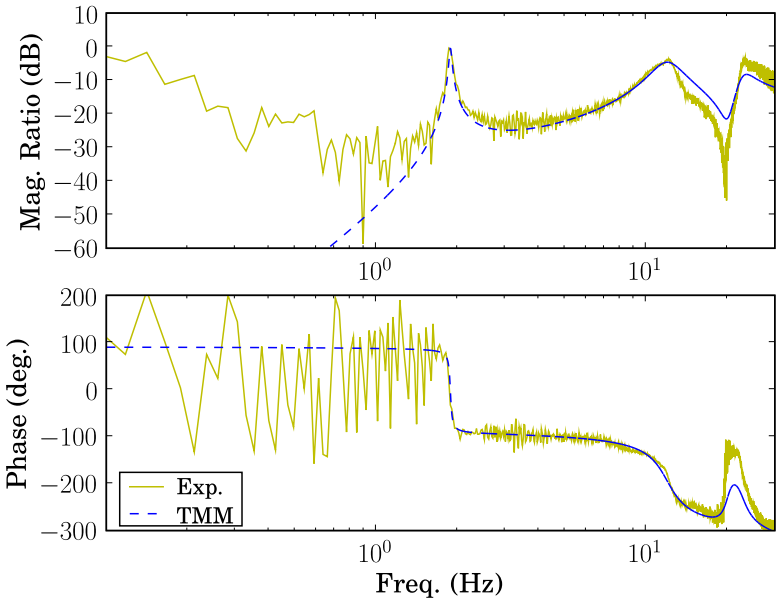

Figure 25: Open-loop flexible base Bode plot $\ddot{x}/v$ comparing experimental data to the TMM model after curve fitting.

Note that in all the Bode plots involving acceleration, the discrepancies between model and experiment below 1.5Hz result from the combination of signal noise and the fact that the system is stiff at low frequencies. The system simply does not vibrate very much at frequencies below 1Hz. This is compounded by the fact that the accelerometers are not as sensitive at low frequencies. As a result, measurements at these frequencies are contaminated by noise. This fact is reflected in Figure 26 which shows the FFT's of $\ddot{x}$ and v along with their coherence. While the magnitude of the FFT $\ddot{x}$ is small at frequencies below 1.5Hz, it is not perfectly 0. Noise on this signal causes poor coherence at these frequencies.

It could be argued that because the coherence is poor below 1.5 Hz, the plot should be cutoff and this portion not shown. However, this frequency range does contain valuable information for the actuator Bode plot (θ/v of Figure 24) as demonstrated by the FFT and coherence plots of Figure 27. The θ and $\ddot{x}$ Bode plots are shown with the same frequency limits to avoid confusion and because they are coupled. Together they give a full picture of the response of the system.

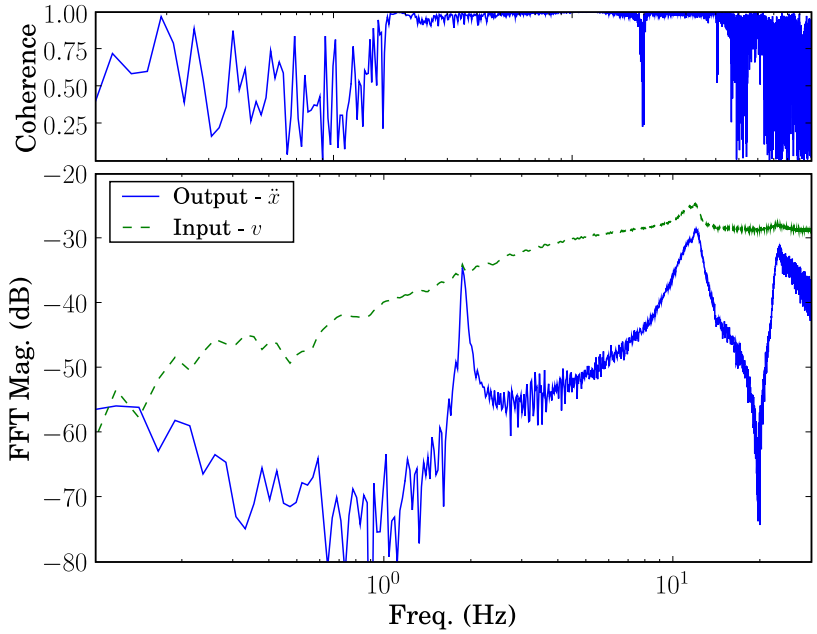

Figure 26: A plot of the FFT's of $\ddot{x}$ and v which are used to calculate the Bode plot of Figure 25, along with the coherence between the two signals.

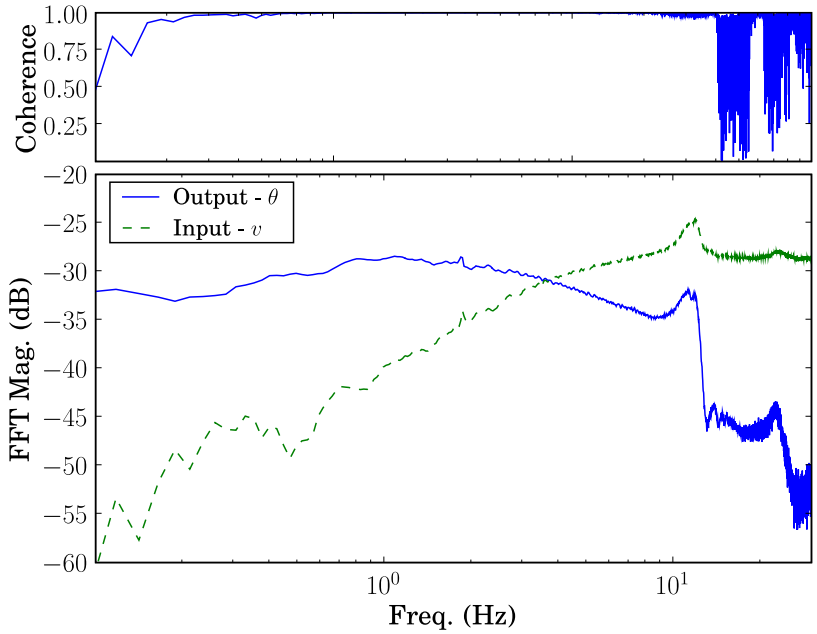

Figure 27: A plot of the FFT's of θ and v which are used to calculate the Bode plot of Figure 24, along with the coherence between the two signals.

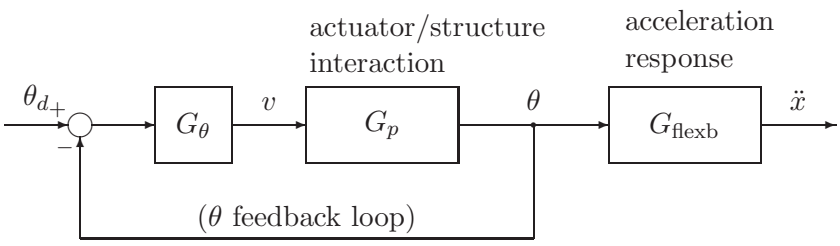

Figure 28: Block diagram of the system with position control (θ feedback).

3.4.2 θ Feedback Modeling

Once the system parameters are determined, the response of the system with θ feedback is modeled in two ways. The transfer matrix method is used directly and as a verification the open-loop transfer functions from the TMM are used to calculated the closed-loop transfer functions based on block diagram algebra.

The TMM model for the system with θ feedback will use $\mathbf{U}_{\text{cl}}$ from equation (63) and the full model will take the form

$$\mathbf{z}_{\text{tip}} = \mathbf{U}_{\text{link2}}\,\mathbf{U}_{\text{cl}}\,\mathbf{U}_{\text{link1}}\,\mathbf{U}_{\text{joint1}}\,\mathbf{U}_{\text{link0}}\,\mathbf{U}_{\text{beam}}\,\mathbf{U}_{\text{basespring}}\,\mathbf{z}_{\text{base}} \tag{102}$$

The block diagram for the θ feedback system is shown in Figure 28. The transfer function verification is based on using the output of the open-loop TMM models to get numeric values for $G_p(s)$ and $G_{\text{flexb}}(s)$. The TMM values from Figure 24 form a vector representing $G_p(s)$. $G_{\text{flexb}}(s)$ can be found by taking the TMM values from Figure 25 and dividing by $G_p(s)$, element by element.

Once $G_p(s)$ and $G_{\text{flexb}}(s)$ are known, determining the closed-loop response using transfer function analysis is straight forward:

$$\frac{\theta}{\theta_d} = \frac{G_\theta G_p}{1 + G_\theta G_p} \tag{103}$$

and

$$\frac{\ddot{x}}{\theta_d} = \left(\frac{G_\theta G_p}{1 + G_\theta G_p}\right) G_{\text{flexb}} \tag{104}$$

The agreement between TMM and transfer function models in Figures 29 and 30 shows that $\mathbf{U}_{\text{cl}}$ from equation (63) is in fact representing the closed-loop control depicted in the block

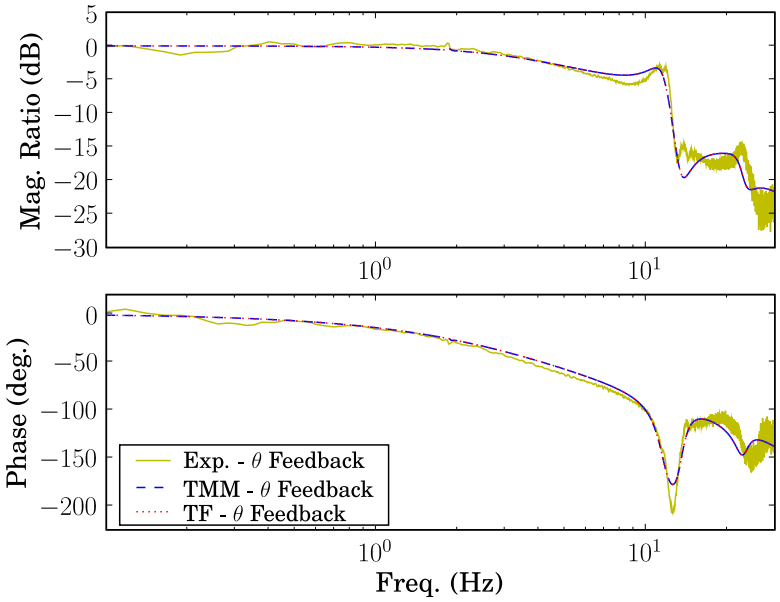

Figure 29: Closed-loop actuator Bode plot θ/θ_d for the system with θ feedback comparing experimental data to the transfer matrix model and a transfer function model.

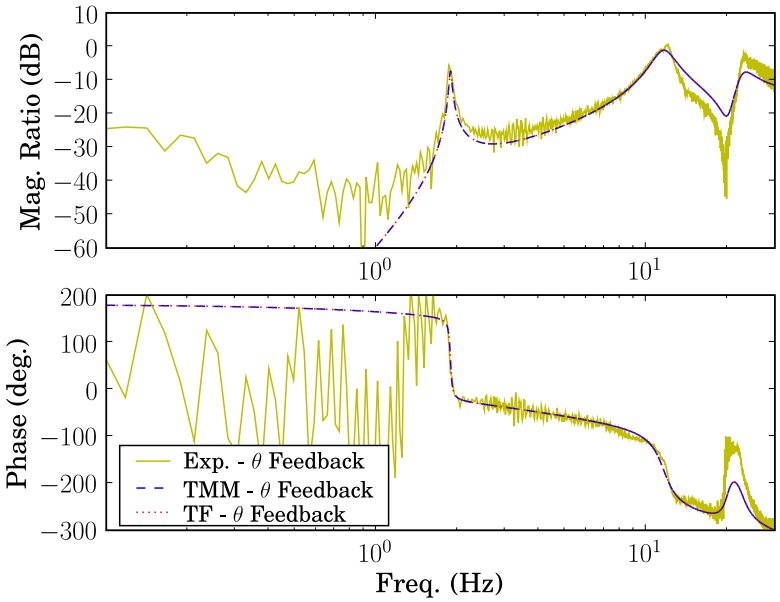

Figure 30: Closed-loop flexible base Bode plot $\ddot{x}/\theta_d$ for the system with θ feedback comparing experimental data to the transfer matrix model and a transfer function model.

diagram of Figure 28. The close agreement between models and experiment in Figures 29 and 30 shows that the models accurately represent the physical system, further verifying the approach. For this analysis, $G_\theta = 1$. Note that the transfer function analysis would not be possible without the open-loop transfer matrix model providing $G_p(s)$ and $G_{\text{flexb}}(s)$ as starting points for the analysis.

3.4.3 Acceleration Feedback

Vibration suppression is achieved by feeding back the acceleration at the end of the cantilever beam into the controller for joint 2, as shown previously in the block diagram of Figure 21. The TMM model of this system will utilize $\mathbf{U}_{\text{acc}}$ defined in equation (100) and will take the form of equation (84).

The transfer function model will again utilize $G_p(s)$ and $G_{\text{flexb}}(s)$ determined from the open-loop TMM model. The θ feedback loop in Figure 21 can be replaced by the equivalent closed-loop transfer function

$$G_{\text{cl}} = \frac{G_\theta G_p}{1 + G_\theta G_p} \tag{105}$$

The block diagram for the system would then look like that in Figure 31. The closed-loop transfer functions with vibration suppression would be

$$\frac{\ddot{x}}{\hat{\theta}_d} = \frac{G_{\text{cl}} G_{\text{flexb}}}{1 + G_{\text{cl}} G_{\text{flexb}} G_a} \tag{106}$$

and

$$\frac{\theta}{\hat{\theta}_d} = \frac{\ddot{x}}{\hat{\theta}_d} \frac{1}{G_{\text{flexb}}} \tag{107}$$

The acceleration feedback controller is

$$G_a = \frac{K_a \, \omega_c^2}{s^2 + 2\,\zeta\,\omega_c\, s + \omega_c^2} \tag{108}$$

where $K_a = 18$, $\zeta = 0.707$, $\omega = 2\pi f_c$ (rad/sec), and $f_c = 2.0$ Hz. The design of G_a and G_θ is discussed extensively in chapter 5. Looking at Figures 32 and 33, the agreement between the TMM and transfer function models shows that $\mathbf{U}_{\text{acc}}$ is correctly modeling the acceleration feedback loop, while the agreement between models and experiment show that the mathematical model accurately captures the real system dynamics.

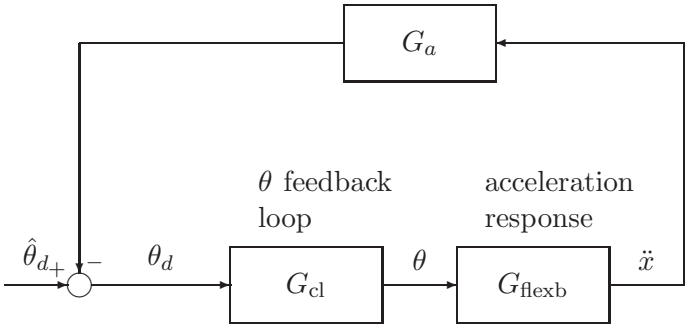

Figure 31: Block diagram for the system with θ and $\ddot{x}$ feedback (vibration suppression). Note that the θ feedback loop has been replaced by the equivalent closed-loop transfer function G_{cl}.

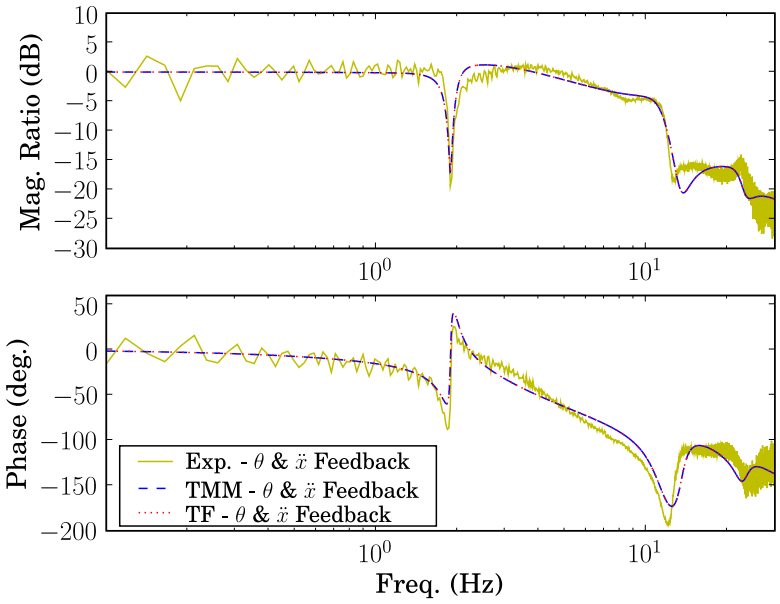

Figure 32: Closed-loop actuator Bode plot $\theta/\hat{\theta}_d$ for the system with θ and $\ddot{x}$ feedback (vibration suppression) comparing experimental data to the transfer matrix model and a transfer function model.

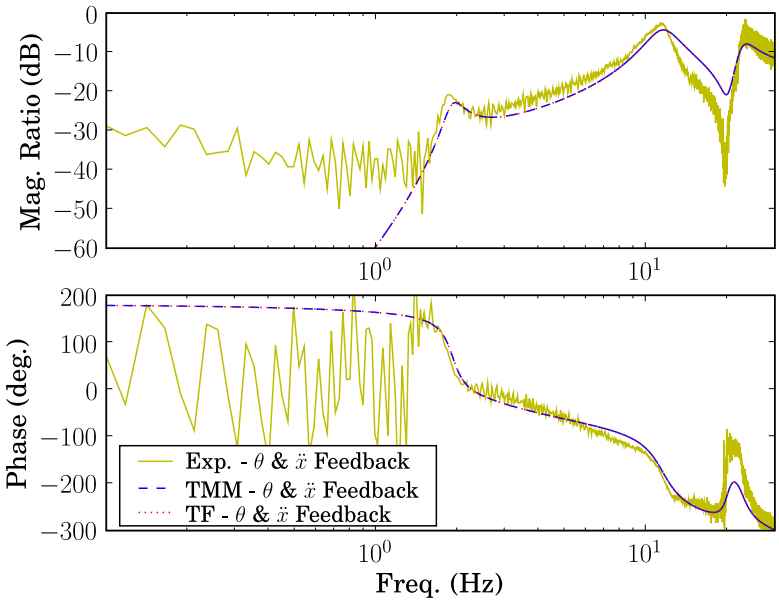

Figure 33: Closed-loop flexible base Bode plot $\ddot{x}/\hat{\theta}_d$ for the system with θ and $\ddot{x}$ feedback (vibration suppression) comparing experimental data to the transfer matrix model and a transfer function model.

3.4.4 Effectiveness of Vibration Suppression

Figure 34 shows the effectiveness of the vibration suppression scheme by comparing experimental accelerometer Bode plots for a system with only θ feedback to one with θ and $\ddot{x}$ feedback. There is a 15 dB reduction in the acceleration response amplitude at the first natural frequency of the system.

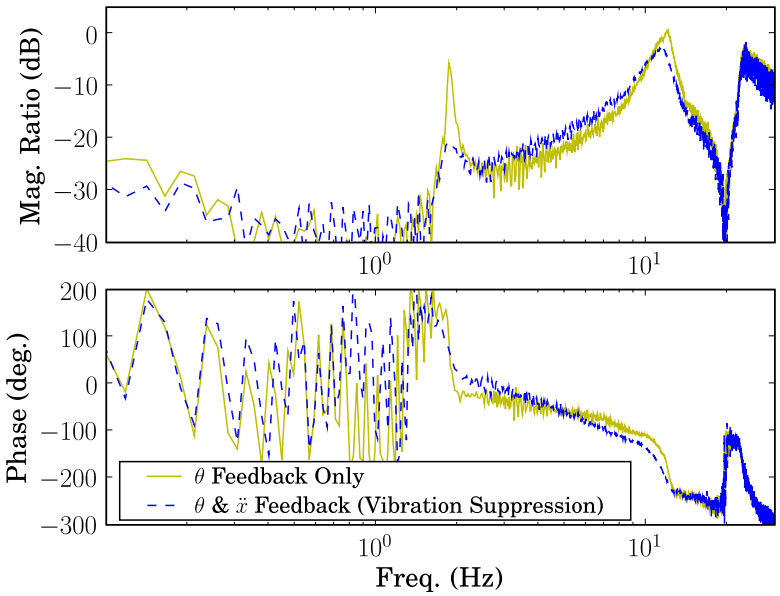

Figure 34: This figure shows the effectiveness of the vibration suppression controller by comparing the experimental flexible base Bode response of a system with only θ feedback ($\ddot{x}/\theta_d$) to one with θ feedback and vibration suppression ($\ddot{x}/\hat{\theta}_d$). There is nearly a 15 dB reduction in amplitude of the response of the first mode of the structure.

CHAPTER IV

MODELING ARBITRARY THREE DIMENSIONAL POSES OF FLEXIBLE ROBOTS

4.1 Introduction

This chapter summarizes the derivation of several three dimensional transfer matrices. The primary contribution of this thesis in this area is proving that the matrices are derived correctly through numerical verification by comparison with FEA. There are several key details that must be considered to get quantitative agreement between the TMM and FEA. Careful attention must be paid to the sign conventions used as the derivations are done about different coordinate axes. All of the transfer matrices must be derived using the same convention for matrix multiplication order. One must also keep track of which states are involved in the different derivations.

4.2 Beam Derivations

A three dimensional beam transfer matrix will be 12×12 and could be thought of as a block diagonal matrix

$$\mathbf{z}_L = \begin{bmatrix} \mathbf{AT}_x & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_y & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{B}_z \end{bmatrix} \mathbf{z}_0 \tag{109}$$

where $\mathbf{AT}_x$ represents the axial and torsional vibration of the x-axis and $\mathbf{B}_y$ and $\mathbf{B}_z$ model bending about the y and z-axes. A sketch of a beam element labeling these axes is shown in Figure 35.

4.2.1 Bending

The bending transfer matrices $\mathbf{B}_y$ and $\mathbf{B}_z$ can be derived from 4 state equations in the spatial variable x.

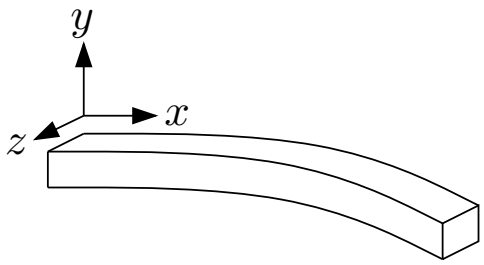

Figure 35: Sketch of a 3D beam element labeling the axes.

For $\mathbf{B}_y$,

$$\theta_y = -\frac{dw_z}{dx} \tag{110}$$

$$M_y = -EI_y\frac{d^2w_z}{dx^2} \tag{111}$$

$$V_z = \frac{\partial M_y}{\partial x} \tag{112}$$

$$\frac{\partial V_z}{\partial x} = \mu\frac{\partial^2 w_z}{\partial t^2} \tag{113}$$

where w is transverse displacement, θ is bending slope, M is bending moment, and V is shear force. EI is the bending stiffness of the beam and μ is the mass per unit length. x is the spatial variable along the length of the beam and t is time.

For $\mathbf{B}_z$,

$$\theta_z = \frac{dw_y}{dx} \tag{114}$$

$$M_z = EI_z\frac{d^2w_y}{dx^2} \tag{115}$$

$$V_y = -\frac{\partial M_z}{\partial x} \tag{116}$$

$$\frac{\partial V_y}{\partial x} = \mu\frac{\partial^2 w_y}{\partial t^2} \tag{117}$$

Note that equations (110)–(112) differ in sign from equations (114)–(116).

Equations (110), (111), (114), and (115) come from mechanics of materials analysis of a beam in bending. Equations (112), (113), (116), and (117) come from applying Newton's second law to a differential beam element. Equations (110) and (114) come from the combination of radius of curvature analysis and the small angle assumption. Figures 36 and 37 sketch these relationships for bending about the z-axis.

Equations (111) and (115) can be derived based on the relationships between curvature,

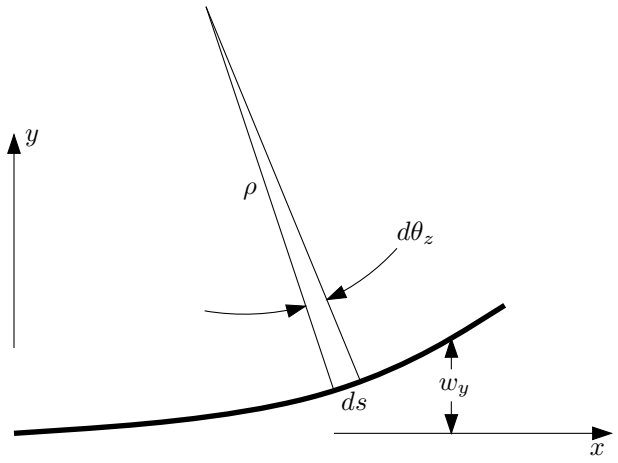

Figure 36: Sketch of a beam undergoing positive angular deflection about the z-axis.

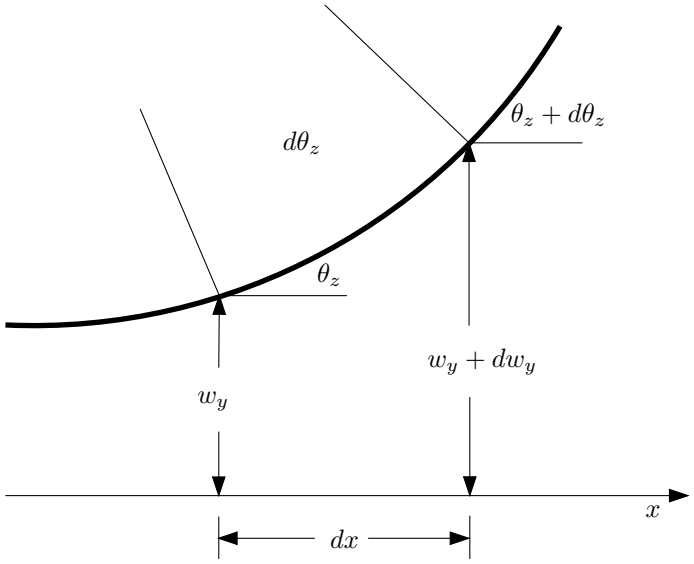

Figure 37: Zooming in on Figure 36.

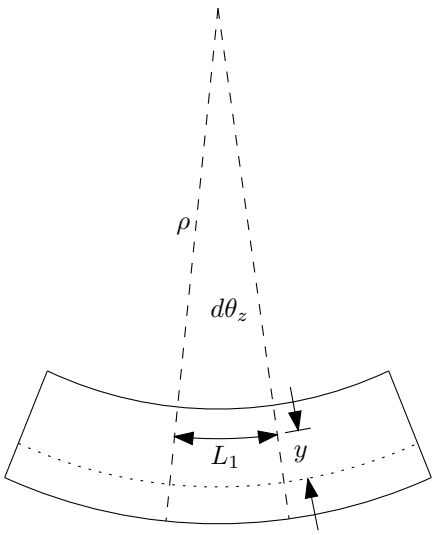

Figure 38: A differential beam element showing the relationship between curvature and strain.

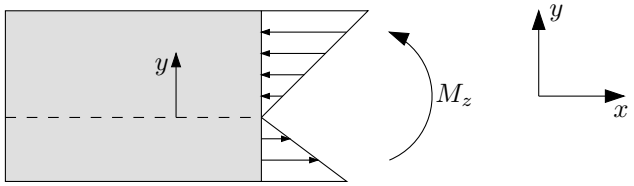

Figure 39: A sketch showing the relationship between stress and moment on a differential element

strain, and bending stress. These relationships are depicted in Figures 38 and 39 for bending about the z-axis.

Equations (112) and (116) come from summing moments on a differential beam element, neglecting rotary inertia and terms of order dx^2. Equations (113) and (117) come from summing forces. A free-body diagram for bending about the z-axis is shown in Figure 40.

Substituting M_z from equation (115) into equation (116) and then substituting the result for V_y into equation (117) gives the partial differential equation of a beam in bending about the z-axis:

$$-EI_z \frac{\partial^4 w_y}{\partial x^4} = \mu \frac{\partial^2 w_y}{\partial t^2} \tag{118}$$

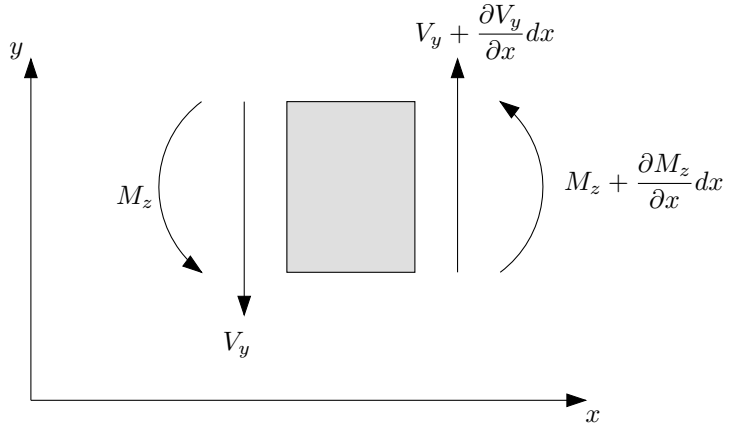

Figure 40: Sketch of the forces and moments acting on a differential beam element bending about the z-axis (vibrating in the xy plane).

Similarly for equations 111-113,

$$-EI_y \frac{\partial^4 w_z}{\partial x^4} = \mu \frac{\partial^2 w_z}{\partial t^2} \tag{119}$$

Equation (118) can be solved by separating time and space variation

$$w_y(x,t) = W_y(x) T_y(t) \tag{120}$$

which leads to the spatial differential equation

$$W_y'''' - \left(\frac{\beta}{L}\right)^4 W_y = 0 \tag{121}$$

where

$$\left(\frac{\beta}{L}\right)^4 = -\omega^2 \frac{\mu}{EI_z} \tag{122}$$

A general solution to equation (121) is given by

$$W_y = A_1 \sin\left(\frac{\beta x}{L}\right) + A_2 \cos\left(\frac{\beta x}{L}\right) + A_3 \sinh\left(\frac{\beta x}{L}\right) + A_4 \cosh\left(\frac{\beta x}{L}\right) \tag{123}$$

Substituting for w_y in equations (114)–(116) from the general solution in equation (123) would give expressions for θ_z, M_z, and V_y in terms of A_i that could be put in matrix form

$$\mathbf{z} = \mathbf{U}(x,\beta)\mathbf{A} \tag{124}$$

where

$$\mathbf{z} = \begin{Bmatrix} w_y \\ \theta_z \\ M_z \\ V_y \end{Bmatrix} \quad \text{and} \quad \mathbf{A} = \begin{Bmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{Bmatrix} \tag{125}$$

Evaluating equation (124) at $x = 0$ and $x = L$ gives

$$\mathbf{z}(0) = \mathbf{U}(0, \beta)\mathbf{A} \tag{126}$$

and

$$\mathbf{z}(L) = \mathbf{U}(L, \beta)\mathbf{A} \tag{127}$$

Solving equation (126) for $\mathbf{A}$ and substituting the result into equation (127) gives

$$\mathbf{z}(L) = \mathbf{U}(L, \beta)\left[\mathbf{U}(0, \beta)\right]^{-1}\mathbf{z}(0) \tag{128}$$

where

$$\mathbf{B}_z = \mathbf{U}(L, \beta)\left[\mathbf{U}(0, \beta)\right]^{-1} \tag{129}$$

defines a transfer matrix for bending about the z-axis. A more detailed explanation of this derivation is included in appendix A.

4.2.2 Axial and Torsional Vibration

4.2.2.1 Axial Vibration

A transfer matrix for axial vibration can be derived following a similar procedure. The states for this transfer matrix will be displacement and force in the x direction (w_x and F_x). The partial differential equation for a rod in axial vibration is [28, p. 318-322]

$$EA\frac{\partial^2 w_x(x,t)}{\partial x^2} = \rho A\frac{\partial^2 w_x(x,t)}{\partial t^2} \tag{130}$$

Using separation of variables leads to the spatial differential equation

$$W_x^{''} - \sigma^2 W_x = 0 \tag{131}$$

A general solution of this equation is

$$W_x = a_1 e^{\sigma x} + a_2 e^{-\sigma x} \tag{132}$$

From solid mechanics it is known that

$$F_x = EAW_x^{'} \tag{133}$$

Substituting for W_x from equation (132) into equation (133) and putting the equations in matrix form produces

$$\mathbf{z}_{\mathrm{Ax}} = \mathbf{U}_{\mathrm{Ax}}(x,\sigma)\mathbf{a} \tag{134}$$

where

$$\mathbf{z}_{\mathrm{Ax}} = \begin{Bmatrix} W_x \\ F_x \end{Bmatrix}, \quad \mathbf{a} = \begin{Bmatrix} a_1 \\ a_2 \end{Bmatrix}, \text{ and } \mathbf{U}_{\mathrm{Ax}}(x,\sigma) = \begin{bmatrix} e^{\sigma x} & e^{-\sigma x} \\ EA\sigma e^{\sigma x} & -EA\sigma e^{-\sigma x} \end{bmatrix} \tag{135}$$

A transfer matrix for axial vibration would be

$$\mathbf{A} = \mathbf{U}_{\mathrm{Ax}}(L,\sigma)\left[\mathbf{U}_{\mathrm{Ax}}(0,\sigma)\right]^{-1} \tag{136}$$

4.2.2.2 Torsional Vibration

The derivation for torsional vibration is nearly identical to that for axial vibration. The states will be angular displacement and moment about the x-axis (θ_x and M_x). The partial differential equation for a rod in torsion is [28, p. 323-327]

$$GJ\frac{\partial^2\theta(x,t)}{\partial x^2} = \rho J\frac{\partial^2\theta(x,t)}{\partial t^2} \tag{137}$$

After separation of variables, the spatial differential equation will be

$$\Theta^{''} - \sigma^2\Theta = 0 \tag{138}$$

with the general solution

$$\Theta = a_1 e^{\sigma x} + a_2 e^{-\sigma x} \tag{139}$$

From solid mechanics it is known that

$$M_x = GJ\Theta^{'} \tag{140}$$

The general solution can again be placed in matrix form:

$$\mathbf{z}_{\mathrm{T}} = \mathbf{U}_{\mathrm{T}}(x,\sigma)\mathbf{a} \tag{141}$$

where

$$\mathbf{z}_{\mathrm{T}} = \begin{Bmatrix} \theta_x \\ M_x \end{Bmatrix}, \quad \mathbf{a} = \begin{Bmatrix} a_1 \\ a_2 \end{Bmatrix}, \text{ and } \mathbf{U}_{\mathrm{T}}(x,\sigma) = \begin{bmatrix} e^{\sigma x} & e^{-\sigma x} \\ GJ\sigma e^{\sigma x} & -GJ\sigma e^{-\sigma x} \end{bmatrix} \tag{142}$$

A transfer matrix for torsional vibration would be

$$\mathbf{T} = \mathbf{U}_{\mathrm{T}}(L,\sigma)\left[\mathbf{U}_{\mathrm{T}}(0,\sigma)\right]^{-1} \tag{143}$$

4.2.2.3 Combined Axial/Torsional Vibration

Equations 136 and 143 can be combined to form a 4×4 axial/torsional transfer matrix for x-axis vibration:

$$\mathbf{AT} = \begin{bmatrix} \mathbf{A}_{11} & 0 & 0 & \mathbf{A}_{12} \\ 0 & \mathbf{T}_{11} & \mathbf{T}_{12} & 0 \\ 0 & \mathbf{T}_{21} & \mathbf{T}_{22} & 0 \\ \mathbf{A}_{21} & 0 & 0 & \mathbf{A}_{22} \end{bmatrix} \tag{144}$$

so that the states are transferred according to

$$\begin{Bmatrix} w_x \\ \theta_x \\ M_x \\ F_x \end{Bmatrix}_L = \mathbf{AT} \begin{Bmatrix} w_x \\ \theta_x \\ M_x \\ F_x \end{Bmatrix}_0 \tag{145}$$

4.2.3 Mixed States

As mentioned previously, the block-diagonal combination of transfer matrices for each axis would give

$$\mathbf{z}_L = \begin{bmatrix} \mathbf{AT}_z & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_y & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{B}_z \end{bmatrix} \mathbf{z}_0 \tag{146}$$

Thinking of the beam matrix in this block diagonal form assumes the following state vectors for the lower 8×8

$$\begin{Bmatrix} w_z \\ \theta_y \\ M_y \\ V_z \\ w_y \\ \theta_z \\ M_z \\ V_y \end{Bmatrix}_R = \begin{bmatrix} \mathbf{B}_{y11} & \mathbf{B}_{y12} & \mathbf{B}_{y13} & \mathbf{B}_{y14} & 0 & 0 & 0 & 0 \\ \mathbf{B}_{y21} & \mathbf{B}_{y22} & \mathbf{B}_{y23} & \mathbf{B}_{y24} & 0 & 0 & 0 & 0 \\ \mathbf{B}_{y31} & \mathbf{B}_{y32} & \mathbf{B}_{y33} & \mathbf{B}_{y34} & 0 & 0 & 0 & 0 \\ \mathbf{B}_{y41} & \mathbf{B}_{y42} & \mathbf{B}_{y43} & \mathbf{B}_{y44} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{B}_{z11} & \mathbf{B}_{z12} & \mathbf{B}_{z13} & \mathbf{B}_{z14} \\ 0 & 0 & 0 & 0 & \mathbf{B}_{z21} & \mathbf{B}_{z22} & \mathbf{B}_{z23} & \mathbf{B}_{z24} \\ 0 & 0 & 0 & 0 & \mathbf{B}_{z31} & \mathbf{B}_{z32} & \mathbf{B}_{z33} & \mathbf{B}_{z34} \\ 0 & 0 & 0 & 0 & \mathbf{B}_{z41} & \mathbf{B}_{z42} & \mathbf{B}_{z43} & \mathbf{B}_{z44} \end{bmatrix} \begin{Bmatrix} w_z \\ \theta_y \\ M_y \\ V_z \\ w_y \\ \theta_z \\ M_z \\ V_y \end{Bmatrix}_L \tag{147}$$

Note that the y and z states are intermixed. In order to make the comparison with FEA and the construction of rotation matrices easier, the states will be separated into x, y, and z blocks. To accomplish this, rows 1 and 4 must be swapped with 5 and 8 and columns 1 and 4 swapped with 5 and 8:

$$\begin{Bmatrix} w_y \\ \theta_y \\ M_y \\ V_y \\ w_z \\ \theta_z \\ M_z \\ V_z \end{Bmatrix}_R = \begin{bmatrix} \mathbf{B}_{z11} & 0 & 0 & \mathbf{B}_{z14} & 0 & \mathbf{B}_{z12} & \mathbf{B}_{z13} & 0 \\ 0 & \mathbf{B}_{y22} & \mathbf{B}_{y23} & 0 & \mathbf{B}_{y21} & 0 & 0 & \mathbf{B}_{y24} \\ 0 & \mathbf{B}_{y32} & \mathbf{B}_{y33} & 0 & \mathbf{B}_{y31} & 0 & 0 & \mathbf{B}_{y34} \\ \mathbf{B}_{z41} & 0 & 0 & \mathbf{B}_{z44} & 0 & \mathbf{B}_{z42} & \mathbf{B}_{z43} & 0 \\ 0 & \mathbf{B}_{y12} & \mathbf{B}_{y13} & 0 & \mathbf{B}_{y11} & 0 & 0 & \mathbf{B}_{y14} \\ \mathbf{B}_{z21} & 0 & 0 & \mathbf{B}_{z24} & 0 & \mathbf{B}_{z22} & \mathbf{B}_{z23} & 0 \\ \mathbf{B}_{z31} & 0 & 0 & \mathbf{B}_{z34} & 0 & \mathbf{B}_{z32} & \mathbf{B}_{z33} & 0 \\ 0 & \mathbf{B}_{y42} & \mathbf{B}_{y43} & 0 & \mathbf{B}_{y41} & 0 & 0 & \mathbf{B}_{y44} \end{bmatrix} \begin{Bmatrix} w_y \\ \theta_y \\ M_y \\ V_y \\ w_z \\ \theta_z \\ M_z \\ V_z \end{Bmatrix}_L \tag{148}$$

4.2.4 Final Beam Transfer Matrix

Defining

$$\mathbf{B}_{8\times 8} = \begin{bmatrix} \mathbf{B}_{z11} & 0 & 0 & \mathbf{B}_{z14} & 0 & \mathbf{B}_{z12} & \mathbf{B}_{z13} & 0 \\ 0 & \mathbf{B}_{y22} & \mathbf{B}_{y23} & 0 & \mathbf{B}_{y21} & 0 & 0 & \mathbf{B}_{y24} \\ 0 & \mathbf{B}_{y32} & \mathbf{B}_{y33} & 0 & \mathbf{B}_{y31} & 0 & 0 & \mathbf{B}_{y34} \\ \mathbf{B}_{z41} & 0 & 0 & \mathbf{B}_{z44} & 0 & \mathbf{B}_{z42} & \mathbf{B}_{z43} & 0 \\ 0 & \mathbf{B}_{y12} & \mathbf{B}_{y13} & 0 & \mathbf{B}_{y11} & 0 & 0 & \mathbf{B}_{y14} \\ \mathbf{B}_{z21} & 0 & 0 & \mathbf{B}_{z24} & 0 & \mathbf{B}_{z22} & \mathbf{B}_{z23} & 0 \\ \mathbf{B}_{z31} & 0 & 0 & \mathbf{B}_{z34} & 0 & \mathbf{B}_{z32} & \mathbf{B}_{z33} & 0 \\ 0 & \mathbf{B}_{y42} & \mathbf{B}_{y43} & 0 & \mathbf{B}_{y41} & 0 & 0 & \mathbf{B}_{y44} \end{bmatrix} \tag{149}$$

allows the final beam transfer matrix to be written as

$$\begin{Bmatrix} w_x \\ \theta_x \\ M_x \\ F_x \\ w_y \\ \theta_y \\ M_y \\ V_y \\ w_z \\ \theta_z \\ M_z \\ V_z \end{Bmatrix}_L = \begin{bmatrix} \mathbf{AT} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & & \mathbf{B}_{8\times 8} \\ \mathbf{0} & & \end{bmatrix} \begin{Bmatrix} w_x \\ \theta_x \\ M_x \\ F_x \\ w_y \\ \theta_y \\ M_y \\ V_y \\ w_z \\ \theta_z \\ M_z \\ V_z \end{Bmatrix}_0 \tag{150}$$

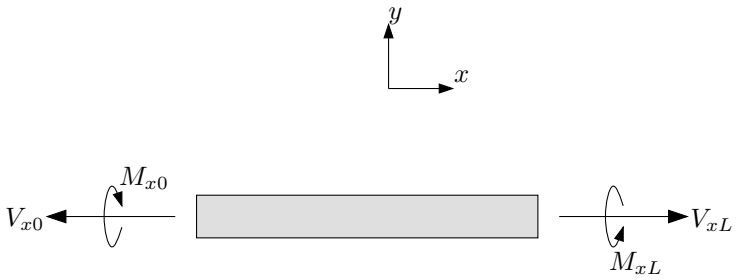

Figure 41: Free-body diagram for a rigid body rotating about and center of mass displacing along the x-axis

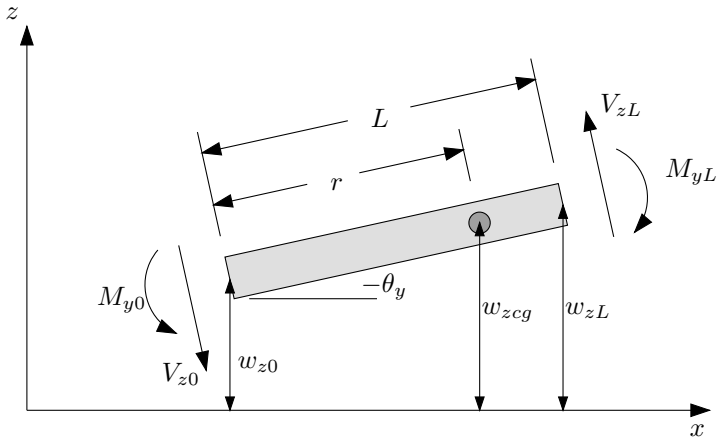

Figure 42: Free-body diagram for a rigid body rotating about the y-axis and center of mass displacing in the z direction.

4.3 Rigid Body Matrix

Much like the beam derivation, the transfer matrix for a rigid body could be broken into 3 sub-matrices:

$$\mathbf{z}_L = \begin{bmatrix} \mathbf{R}_x & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{R}_y & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{R}_z \end{bmatrix} \mathbf{z}_0 \tag{151}$$

The rigid transfer matrix for each axis will depend on 4 equations: the sum of the forces, the sum of the moments, and equations for the displacement and rotation that come from assuming the element is rigid. Free-body diagrams for each axis are shown in Figures 41–43.

For all three axes, the rigidity of the element means that the rotation does not change across the element

$$\theta_L = \theta_0 \tag{152}$$

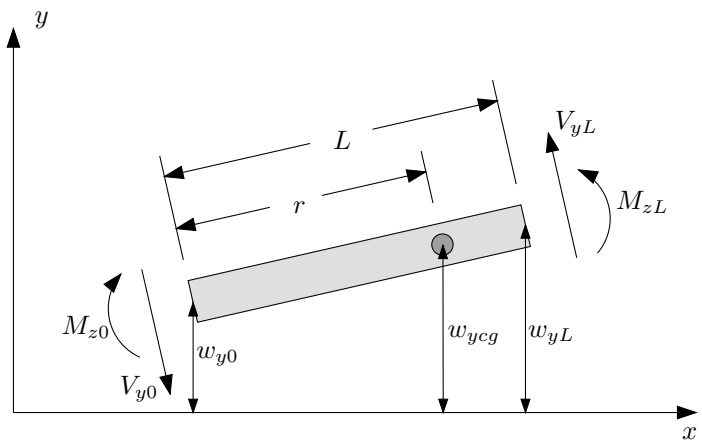

Figure 43: Free-body diagram for a rigid body rotating about the z-axis and center of mass displacing in the y direction.

For the x-axis, the displacement is also constant across the element:

$$w_{xL} = w_{x0} \tag{153}$$

For the y and z-axes, the rotation affects the displacement:

$$w_{yL} = w_{y0} + \theta_z L \tag{154}$$

$$w_{zL} = w_{z0} - \theta_y L \tag{155}$$

Note that equations (154) and (155) assume small angles of rotation.

The final transfer matrix for each axis is

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & s^2 I_x & 1 & 0 \\ ms^2 & 0 & 0 & 1 \end{bmatrix} \tag{156}$$

$$\mathbf{R}_y = \begin{bmatrix} 1 & -L & 0 & 0 \\ 0 & 1 & 0 & 0 \\ ms^2(L-r) & s^2 I_y - ms^2 r(L-r) & 1 & L \\ ms^2 & -ms^2 r & 0 & 1 \end{bmatrix} \tag{157}$$

$$\mathbf{R}_z = \begin{bmatrix} 1 & L & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -ms^2(L-r) & s^2 I_z - ms^2 r(L-r) & 1 & -L \\ ms^2 & ms^2 r & 0 & 1 \end{bmatrix} \tag{158}$$

where the corresponding state vectors are

$$\mathbf{z}_{\mathrm{x}} = \begin{Bmatrix} w_x \\ \theta_x \\ M_x \\ V_x \end{Bmatrix} \tag{159}$$

$$\mathbf{z}_{\mathrm{y}} = \begin{Bmatrix} w_z \\ \theta_y \\ M_y \\ V_z \end{Bmatrix} \tag{160}$$

$$\mathbf{z}_{\mathrm{z}} = \begin{Bmatrix} w_y \\ \theta_z \\ M_z \\ V_y \end{Bmatrix} \tag{161}$$

4.3.1 Final Rigid Body Matrix/Mixed States

A block diagonal rigid body transfer matrix would take the form

$$\mathbf{z}_L = \begin{bmatrix} \mathbf{R}_x & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{R}_y & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{R}_z \end{bmatrix} \mathbf{z}_0 \tag{162}$$

The rows and columns of the lower 8×8 of this matrix need to be swapped just like the beam matrix in equation (149). The final form of the rigid body 12×12 transfer matrix

would then be

$$\begin{Bmatrix} w_x \\ \theta_x \\ M_x \\ F_x \\ w_y \\ \theta_y \\ M_y \\ V_y \\ w_z \\ \theta_z \\ M_z \\ V_z \end{Bmatrix}_L = \begin{bmatrix} \mathbf{R}_x & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & & \mathbf{R}_{8\times 8} \\ \mathbf{0} & & \end{bmatrix} \begin{Bmatrix} w_x \\ \theta_x \\ M_x \\ F_x \\ w_y \\ \theta_y \\ M_y \\ V_y \\ w_z \\ \theta_z \\ M_z \\ V_z \end{Bmatrix}_0 \tag{163}$$

4.4 Rotation Matrix

Transfer matrices for joints that allow the robot to be put in arbitrary poses will be 12×12 rotation matrices. These matrices are conceptually similar to standard 3×3 rotation matrices, except that each element of the 3×3 matrix is replaced by itself times a 4×4 identity matrix. For example, consider rotation about the z-axis. The standard 3×3 rotation matrix is

$$\mathbf{R}_z(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{164}$$

where α is the angle of rotation about the z-axis. For a 12×12 transfer matrix, each element of $\mathbf{R}_z$ actually represents a 4×4 matrix with its value on the diagonal:

$$\mathbf{R}_{12z}(\alpha) = \begin{bmatrix} c_\alpha & 0 & 0 & 0 & -s_\alpha & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & c_\alpha & 0 & 0 & 0 & -s_\alpha & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & c_\alpha & 0 & 0 & 0 & -s_\alpha & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & c_\alpha & 0 & 0 & 0 & -s_\alpha & 0 & 0 & 0 & 0 \\ s_\alpha & 0 & 0 & 0 & c_\alpha & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & s_\alpha & 0 & 0 & 0 & c_\alpha & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & s_\alpha & 0 & 0 & 0 & c_\alpha & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & s_\alpha & 0 & 0 & 0 & c_\alpha & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{165}$$

where $c_\alpha = \cos\alpha$ and $s_\alpha = \sin\alpha$ have been used to make this matrix fit on the page.

4.5 Numerical Verification

The combination of a beam matrix, a rotation matrix, and a rigid body matrix will be tested in this section. This combination will be used to model an L-shaped structure with a cantilevered beam along the x-axis and a rigid body going off at 90°. The rigid body will have a free boundary condition on its far end. Natural frequencies and mode shapes from the TMM and FEA will be compared.

4.5.1 TMM Analysis Procedure

The procedure for finding the natural frequencies and mode shapes of the L-shaped structure is as follows. First, find transfer matrices for each element in the system. The element matrices were derived in sections 4.2–4.4. Next, multiply the element transfer matrices to form the system transfer matrix:

$$\mathbf{U}_{\text{sys}} = \mathbf{U}_{\text{rigid}}\,\mathbf{U}_{\text{z}}\,\mathbf{U}_{\text{beam}} \tag{166}$$

where $\mathbf{U}_{\text{rigid}}$ is a rigid body transfer matrix, $\mathbf{U}_{\text{z}}$ is a rotation matrix about the z-axis, and $\mathbf{U}_{\text{beam}}$ is a transfer matrix for a beam element. The system model will be

$$\mathbf{z}_{\text{tip}} = \mathbf{U}_{\text{sys}}\,\mathbf{z}_{\text{base}} \tag{167}$$

and the system boundary conditions require that all displacement and rotation states be zero at the base and all force and moment states be zero at the tip. Partitioning equation (167) appropriately gives

$$\begin{Bmatrix} 0 \\ \vdots \\ 0 \end{Bmatrix} = \mathbf{subU}(s)\hat{\mathbf{z}}_{\text{base}} \tag{168}$$

The natural frequencies are are the values of s that cause $\mathbf{subU}(s)$ to have a null space, so that equation (168) has a non-trivial solution. These values of s will have to be searched for numerically. Once a value of s that causes $\mathbf{subU}(s)$ to have a null space is found, the vector $\hat{\mathbf{z}}_{\text{base}}$ that forms the null space can be found. Section 8.3 discusses the procedure for systems with repeated roots. Once $\hat{\mathbf{z}}_{\text{base}}$ has been found, it can be combined with the known 0 values from the boundary conditions to form $\mathbf{z}_{\text{base}}$.

Once $\mathbf{z}_{\text{base}}$ is known, the state vector at any location in the model can be found. For example, the state vector at the end of the beam will be

$$\mathbf{z}_{\text{beam}} = \mathbf{U}_{\text{beam}}\,\mathbf{z}_{\text{base}} \tag{169}$$

The state vector at the beginning of the rigid link will be

$$\mathbf{z}_{\text{joint}} = \mathbf{U}_{\text{z}}\,\mathbf{U}_{\text{beam}}\,\mathbf{z}_{\text{base}} \tag{170}$$

The state vector at the free end of the structure will be

$$\mathbf{z}_{\text{tip}} = \mathbf{U}_{\text{rigid}}\,\mathbf{U}_{\text{z}}\,\mathbf{U}_{\text{beam}}\,\mathbf{z}_{\text{base}} = \mathbf{U}_{\text{sys}}\,\mathbf{z}_{\text{base}} \tag{171}$$

The TMM calculates the exact transfer of the state vector from one end of the beam to the other without calculating any states along the length of the beam. In order to visualize the displacement along the length of the beam, the beam will need to be broken up into several smaller beams. The states along the length of the beam can then be found from

$$\mathbf{z}_{\text{beam}}(x) = \mathbf{U}_{\text{beam}}(x)\,\mathbf{z}_{\text{base}} \tag{172}$$

where $\mathbf{U}_{\text{beam}}(x)$ is a beam transfer matrix found by substituting x for L in the beam transfer matrices derived in section 4.2, provided that $x \leq L$.

4.5.2 Analysis Details

The properties of the beam in this L-shape structure are as follows:

$$\begin{aligned} EA &= 1.45736e+08 \text{ N} \\ EI_y &= 3.39137e+05 \text{ N}\times\text{m}^2 \\ EI_z &= 5.08706e+05 \text{ N}\times\text{m}^2 \\ GJ &= 2.55774e+05 \text{ N}\times\text{m}^2 \\ \mu &= 5.72815 \text{ kg/m} \\ L &= 4.64820 \text{ m} \end{aligned} \tag{173}$$

The shearing stiffness are set to large values and the moments of inertia of the beam elements are set to small values to emulate an Euler-Bernoulli beam. The actual numeric value for shear stiffness was 1.3e+13 for both the x and y-axes. The moment of inertia for the y and z-axes was 1.33298e–02 and $I_x = I_y + I_z$. The beam was meshed with 50 third order elements.

The rigid mass properties are as follows:

$$\begin{aligned} m &= 2.0 \text{ kg} \\ L &= 0.5 \text{ m} \\ r &= 0.4 \text{ m} \\ I_x &= 0.433333 \text{ kg}\times\text{m}^2 \\ I_y &= 3.28333 \text{ kg}\times\text{m}^2 \\ I_z &= 3.68333e \text{ kg}\times\text{m}^2 \end{aligned}$$

where r is the distance to the center of mass from the base of the rigid link. The FEA analysis was done using the Dymore multi-body code [5] and the full input file is available in appendix B.

4.5.3 Natural Frequencies

A comparison of the natural frequencies from the TMM and FEA is shown in Table 1.

Table 1: Comparison of the first 9 natural frequencies found by the TMM and FEA.

Mode #	FEA (Hz)	TMM(Hz)	% Difference
1	3.0934	3.0937	-0.00823
2	3.7973	3.7976	-0.00922
3	25.794	25.804	-0.0376
4	31.339	31.356	-0.0553
5	53.422	53.482	-0.112
6	75.434	75.495	-0.0801
7	99.203	99.505	-0.304
8	143.73	144.04	-0.22
9	171.25	171.93	-0.392

4.5.4 Mode Shapes

Mode shapes for modes 1, 2, 5, 6, and 7 of the L-shaped structure are shown in Figures 44–53. These modes were chosen as representative of the excellent agreement between TMM and FEA for all the modes listed in Table 1. The modes shapes from FEA and TMM are compared by showing two figures per mode. The first figure is a 2D plot of all 6 states of the beam against the x coordinate of the mesh. In these plots u, v, and w refer to displacements in the x, y, and z directions respectively.

In these 2D plots, the mode shapes from FEA are shown with various line types while the TMM mode shapes are shown with symbols. Each degree of freedom has a unique color and the same color is used for TMM and FEA. Note that the TMM calculates the exact transfer of all states from one end of the beam to the other without calculating the states along the length of the beam. The states at the symbol locations are calculated for display and comparison purposes after the TMM analysis is complete. The values at the symbols are exact and in near perfect agreement with FEA. These comparisons could be done at more locations along the beam if desired. The second figure for each mode is a 3D wireframe generated using the 6 states shown in the other plots.

Note that the amplitudes of the mode shapes from the TMM are scaled to match those of the FEA package chosen for this work (Dymore). Dymore finds the maximum value of

all of the state variables at all locations in the model and scales the mode shape so that that maximum value is 0.2.

Also note that in Figure 46, 48, 50, and 52 there is one state variable whose plot ends in a vertical line segment. This comes from the fact that both ends of the rigid link have the same x coordinate in the undeformed position.

In all of these plots, the results from the TMM and FEA are indistinguishable.

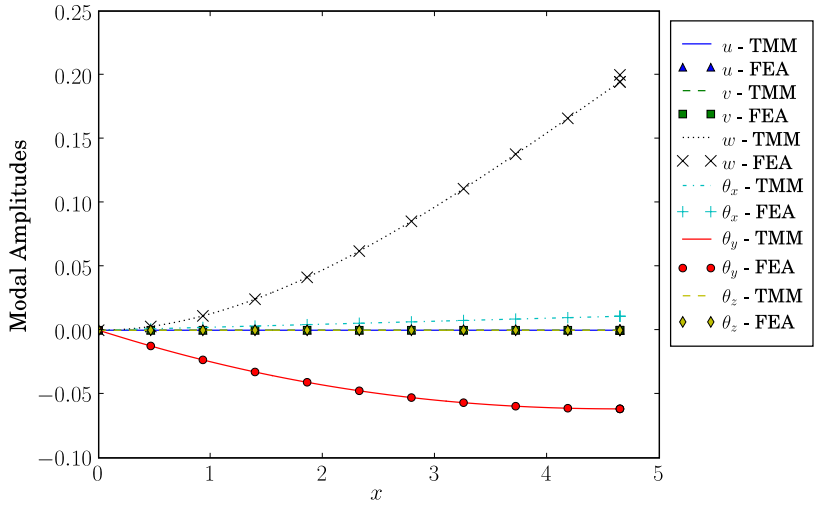

Figure 44: Comparison of TMM and FEA mode shape for mode 1: plotting all 6 states of the structure vs. the x coordinate of the mesh.

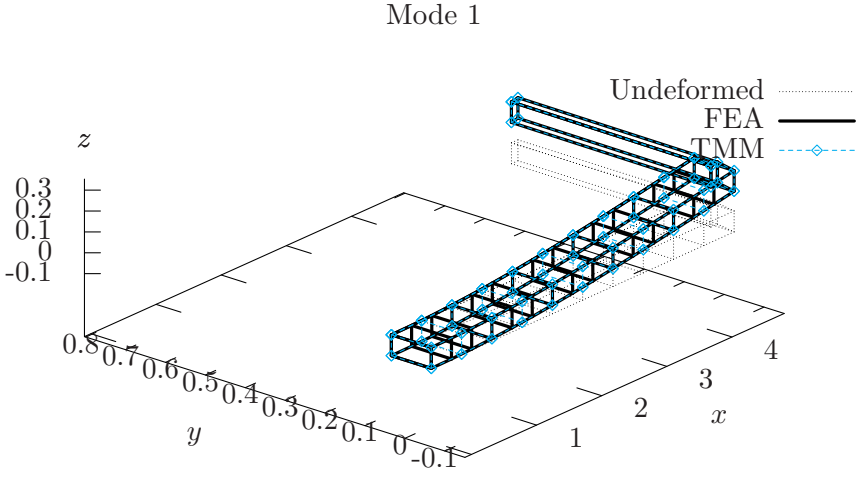

Figure 45: 3D wireframe comparison of TMM and FEA mode shape for mode 1.

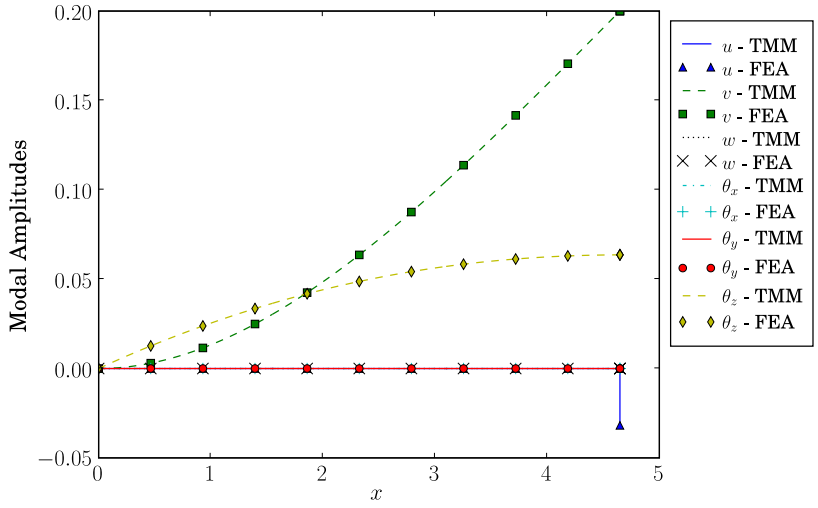

Figure 46: Comparison of TMM and FEA mode shape for mode 2: plotting all 6 states of the structure vs. the x coordinate of the mesh.

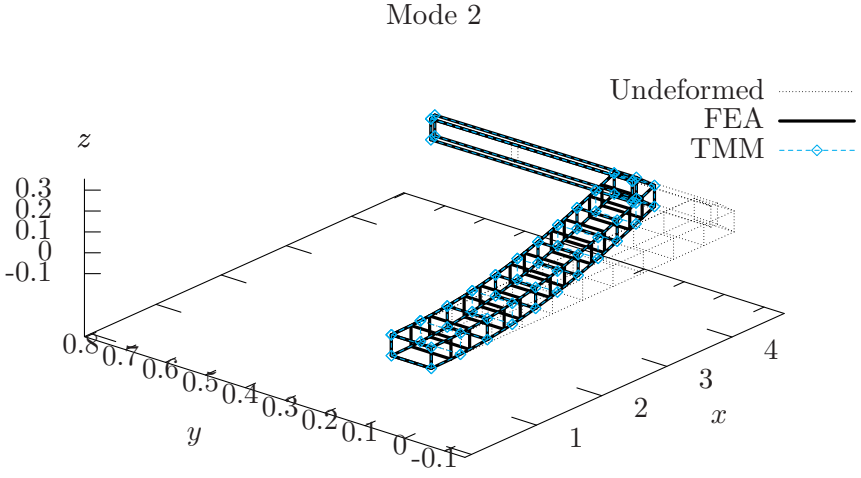

Figure 47: 3D wireframe comparison of TMM and FEA mode shape for mode 2.

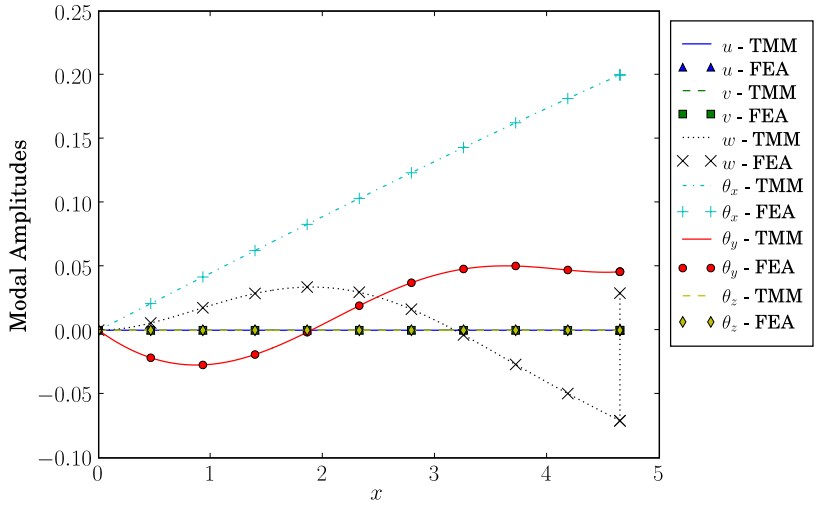

Figure 48: Comparison of TMM and FEA mode shape for mode 5: plotting all 6 states of the structure vs. the x coordinate of the mesh.

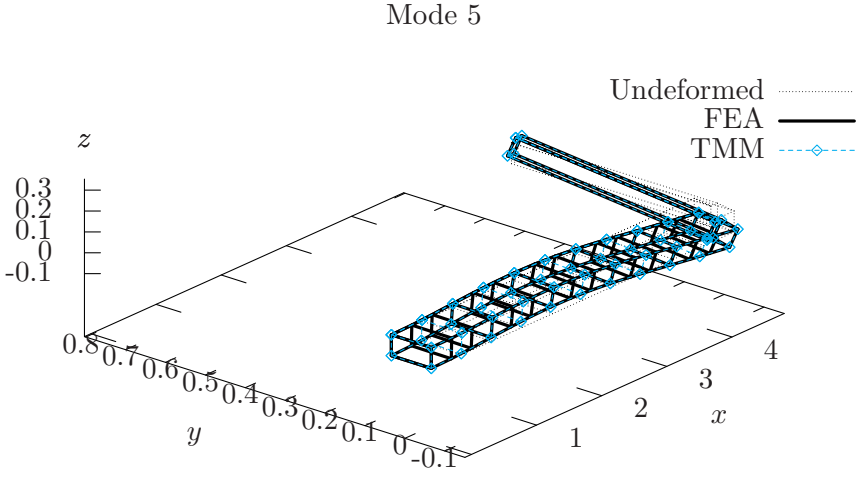

Figure 49: 3D wireframe comparison of TMM and FEA mode shape for mode 5.

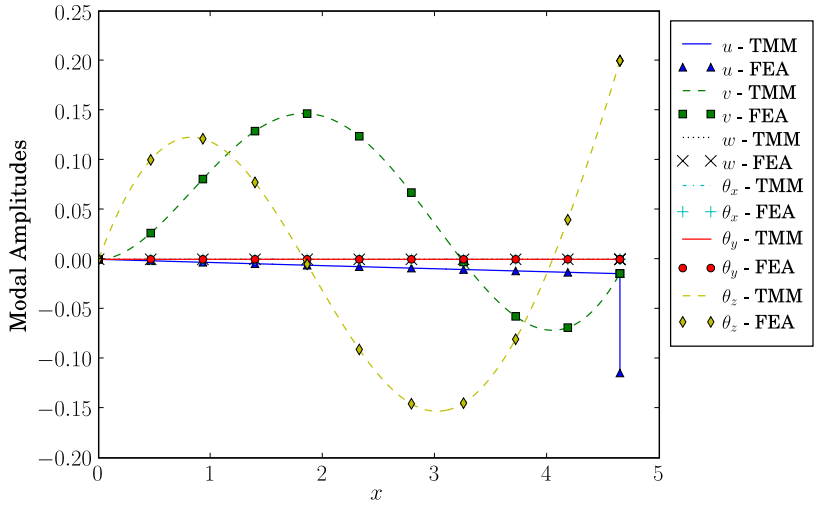

Figure 50: Comparison of TMM and FEA mode shape for mode 6: plotting all 6 states of the structure vs. the x coordinate of the mesh.

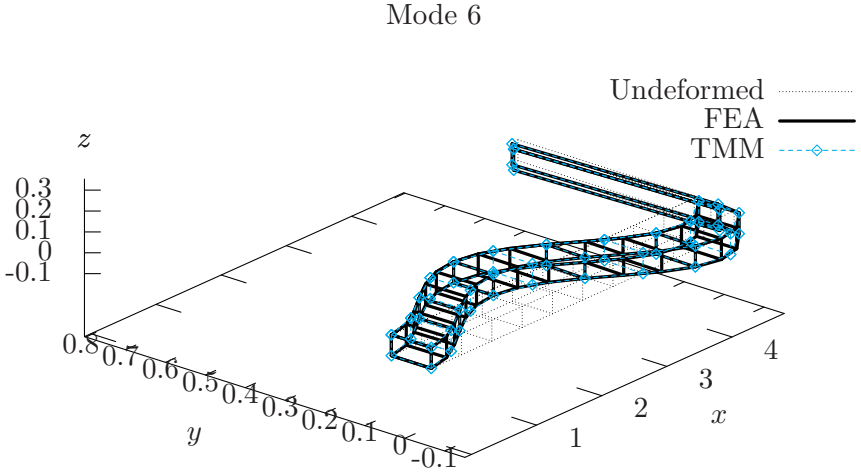

Figure 51: 3D wireframe comparison of TMM and FEA mode shape for mode 6.

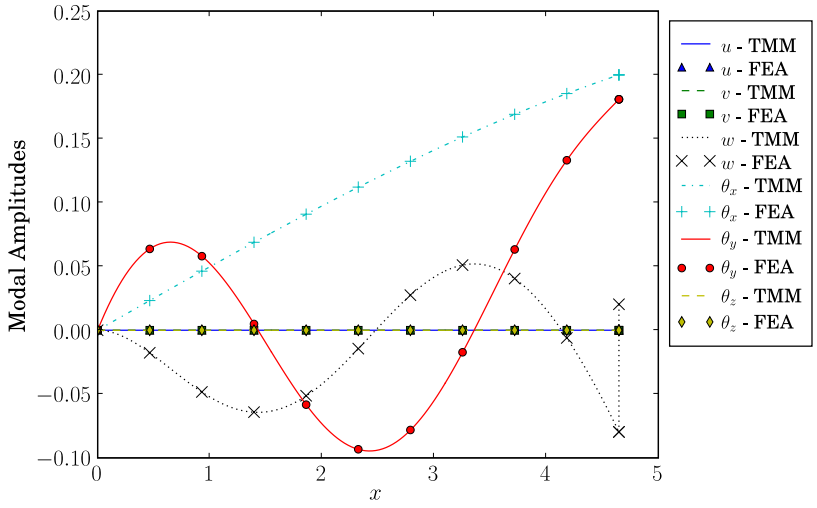

Figure 52: Comparison of TMM and FEA mode shape for mode 7: plotting all 6 states of the structure vs. the x coordinate of the mesh.

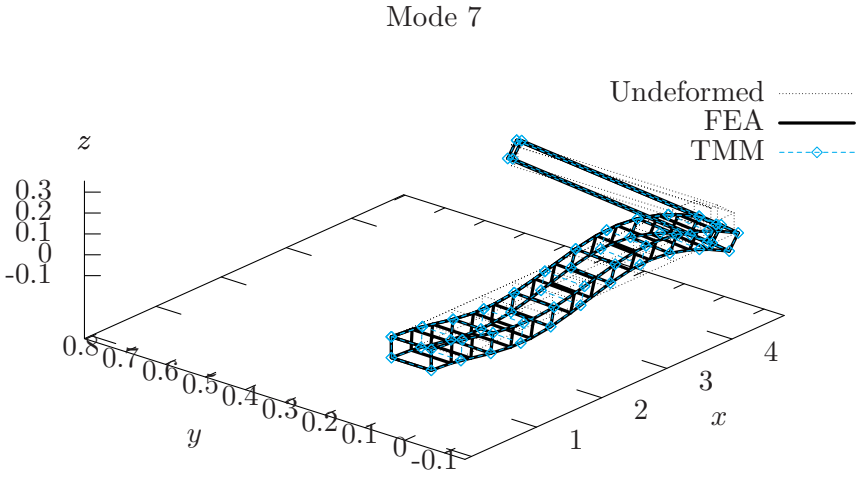

Figure 53: 3D wireframe comparison of TMM and FEA mode shape for mode 7.

CHAPTER V

CONTROL DESIGN USING THE TMM

5.1 Introduction

Chapter 3 demonstrated the ability of the TMM to model the closed-loop response of systems under non-collocated feedback and with multiple feedback loops. This chapter explores how to use such a model for control design. There are at least three viable options. The first is to do compensator design using Bode plots. This approach is usually graphical and iterative. Because the TMM outputs Bode plots so naturally, this approach is very easy to implement. However, it may not work very well for designing complicated controllers and it can be somewhat imprecise compared to more modern design techniques. Section 5.2 presents an approach to automating Bode-based control design by posing it as an optimization problem.

A second approach, created by Book and Majette [9, 41], combines the TMM with state-space control design and pole-placement. This approach uses modal discretization to convert the TMM model to state-space. State-feedback gains are then determined from the discretized model. Iteration may be required if the controller affects the natural frequencies and mode shapes of the system.

Section 5.3 presents a third control design technique that is based on pole-placement design but avoids the modal discretization of the method of Book and Majette.

5.2 Automated Bode Design and Optimization

Figure 54 shows the open-loop block diagram for the system. The control design will follow the approach of previous researchers [18, 37, 23] and use two feedback loops, one for motion control using θ feedback, and one for vibration suppression using $\ddot{x}$ feedback, as shown in the block diagram of Figure 55.

Many approaches to designing controllers for flexible robots design these two feedback

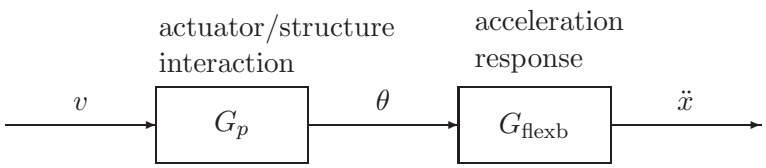

Figure 54: Block diagram of the open-loop system.

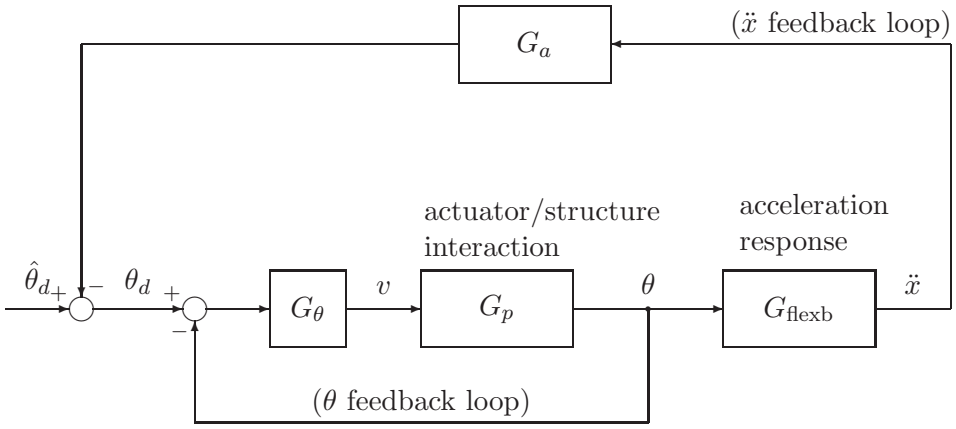

Figure 55: Block diagram of the system with motion control (θ feedback) and vibration suppression ($\ddot{x}$ feedback).

loops separately. The motion control loop is considered "slow" and is designed first. The vibration suppression controller is considered "fast" and is designed second. A singular perturbation approach is used to justify the assumption that these two controllers do not interact with one another [55].

This work has overcome this limitation by giving the TMM the ability to model multiple, non-collocated feedback loops and design these controllers simultaneously. Later in this chapter it will be shown that there is benefit to simultaneously designing these controllers. For now however, it will aid the discussion to follow the same approach and design these two feedback loops separately. The θ feedback loop will be designed first, followed by the $\ddot{x}$ feedback loop.

5.2.1 θ Feedback Design

The system with θ feedback is shown in Figure 56. The goal of θ feedback design is to determine G_θ so that the closed-loop system responds as quickly as possible while maintaining

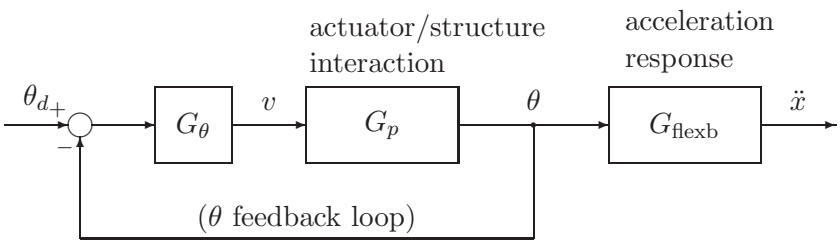

Figure 56: Block diagram of the system with θ feedback.

system stability and avoiding excessive overshoot. A detailed explanation of Bode-based control design can be found in many controls textbooks (see for example [21, Chapter 11]). For this example, it will be sufficient to maximize the crossover frequency while maintaining a minimum phase margin of 60°. The crossover frequency is related to the system bandwidth and the speed of response of the system. Phase margin relates closely to the damping of the system and is a good measure of overshoot and relative stability. Crossover frequency is the frequency where the magnitude of the Bode plot passes through 0 dB. Phase margin is 180° plus the phase at the crossover frequency. When the phase margin is 0°, the system has zero damping and is marginally stable. Phase margin of 60° corresponds to $\zeta = 0.7$ for a simple second-order system. Bode control design is based on using the loop transfer function $G_\theta G_p$ to predict the closed-loop response G_{cl} where

$$G_{\text{cl}} = \frac{G_\theta G_p}{1 + G_\theta G_p} \tag{174}$$

The open-loop Bode plot for the actuator is shown in Figure 57. The crossover frequency is 4.63 Hz and the phase margin is 76.9°. Since the phase margin is greater than the specified minimum, the gain could be increased. Increasing the gain will increase the crossover frequency while decreasing the phase margin. For this example a proportional controller is used so that

$$G_\theta = K_\theta \tag{175}$$

where K_θ is the proportional gain.

h

Figure 58 shows Bode plots for the loop transfer function $G_\theta G_p$ for various values of K_θ. Table 2 shows the corresponding values for the crossover frequencies and phase margins.

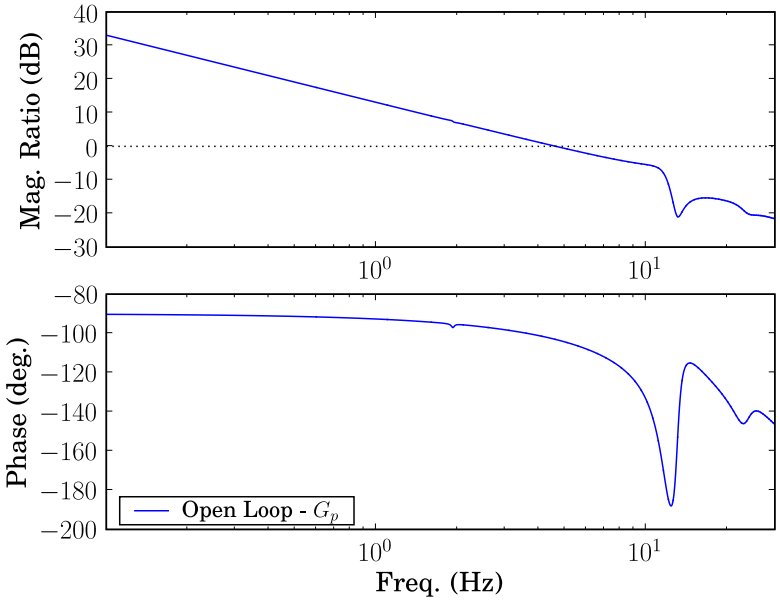

Figure 57: Open-loop Bode plot for the actuator $G_p = \theta/v$.

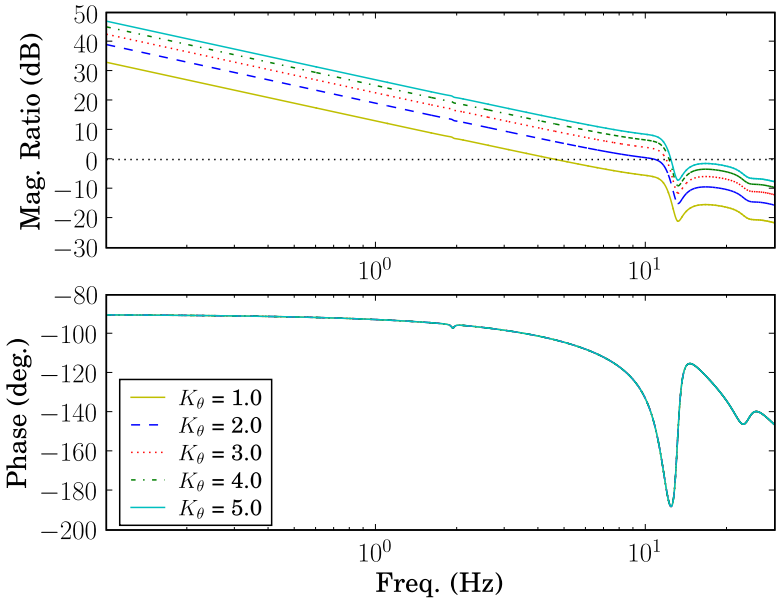

Figure 58: Loop transfer Bode plots $G_\theta G_p$ for various values of K_θ.

Table 2: Crossover frequency and phase margin vs. K_θ.

K_θ	f_c (Hz)	Phase Margin (deg.)
1.0	4.63	76.90
2.0	10.82	32.17
3.0	11.88	1.14
4.0	12.18	-6.17
5.0	12.37	-7.99

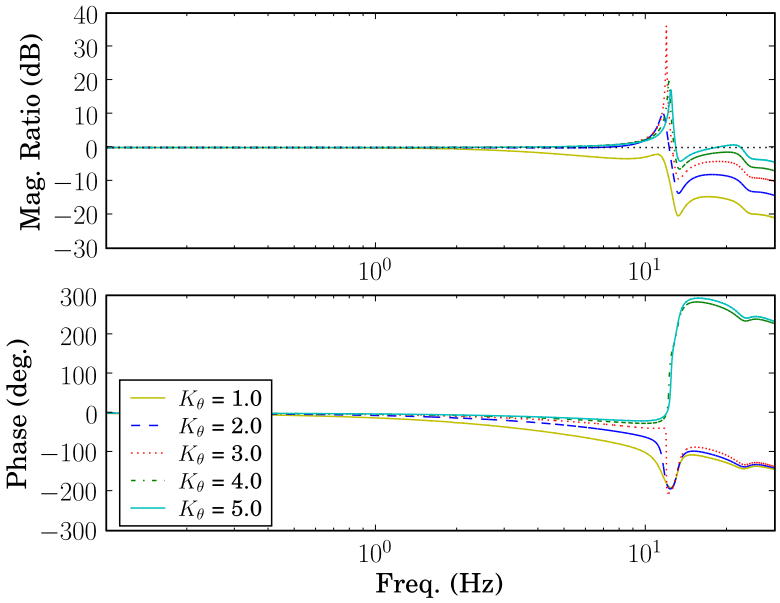

Figure 59: Closed-loop actuator Bode plots G_{cl} for various values of K_θ.

Figure 59 shows the corresponding closed-loop (G_{cl}) Bode plots. Note that G_{cl} is unstable for $K_\theta = 4.0$ or 5.0. The instability can be seen in the negative phase margins of Table 2 and Figure 58 and in the fact that the phase increases rather than decreases near the pole at 11.5 Hz in Figure 59.

The optimal K_θ is found using a cost function where the cost goes down as the crossover frequency moves to the right and there is a penalty when the phase margin is less than 60°. The cost function is:

```
1 def _actcost(Gth):
2     GthGp = CompGthGpBode(Gth)
3     cf, ind = GthGp.CrossoverFreq(f)
4     pm = GthGp.PhaseMargin(f)
5     cost = 100-cf#drive the crossover frequency to the right
```

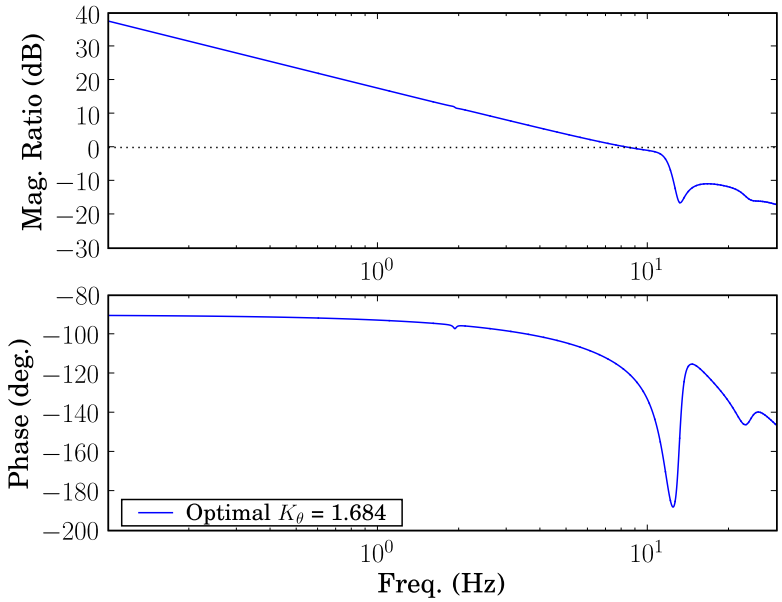

Figure 60: Bode plot of the loop transfer function $G_\theta G_p$ for the optimal $K_\theta = 1.684$.

```
6       if pm<60.0:
7           cost+=(60.0-pm)**2
8       return cost
```

Line 2 calculates the loop transfer function for $G_\theta G_p$. Line 3 finds the crossover frequency and line 4 finds the phase margin for $G_\theta G_p$. Line 5 defines the initial cost as 100 minus the crossover frequency, and lines 6 and 7 apply the penalty if the phase margin is less than 60°. The optimal K_θ is 1.684. Figure 60 shows the Bode plot for the loop transfer function $G_\theta G_p$ for the optimal K_θ and Figure 61 shows the corresponding closed-loop Bode plot G_{cl}.

5.2.2 Acceleration Feedback Design

Now that the θ feedback loop has been designed (G_θ has been specified), the $\ddot{x}$ feedback loop can be designed. The $\ddot{x}$ feedback controller will be designed to reject disturbances. These disturbances will be measured with the accelerometer at the end of SAMII's cantilever beam. Defining the transfer functions

$$G_p = \frac{\theta}{v} \tag{176}$$

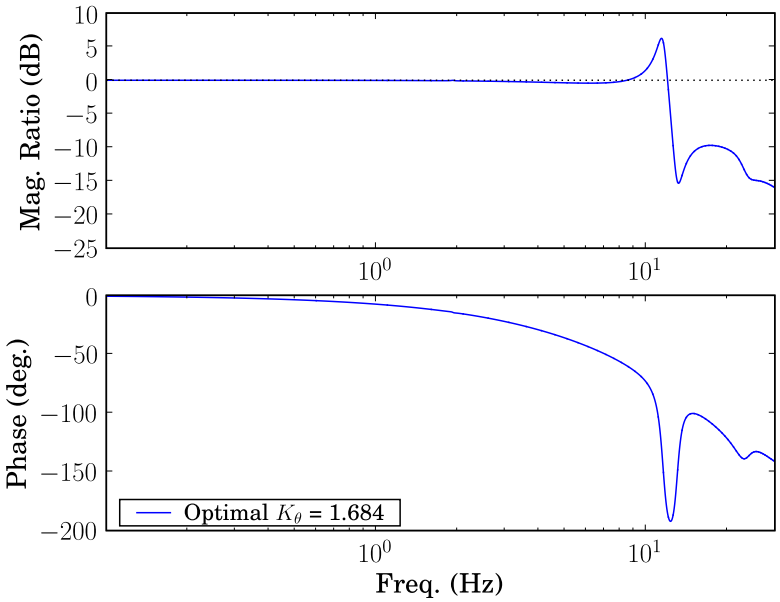

Figure 61: Closed-loop actuator Bode plot $G_{\text{cl}} = \theta/\theta_d$ for the optimal $K_\theta = 1.684$.

and

$$G_{\text{flexb}} = \frac{\ddot{x}}{\theta} \tag{177}$$

allows the block diagram of Figure 55 to be simplified by replacing the θ feedback loop with the closed-loop transfer function

$$G_{\text{cl}} = \frac{G_\theta G_p}{1 + G_\theta G_p} = \frac{\theta}{\theta_d} \tag{178}$$

Figure 62 shows the simplified block diagram along with the disturbance input d. The

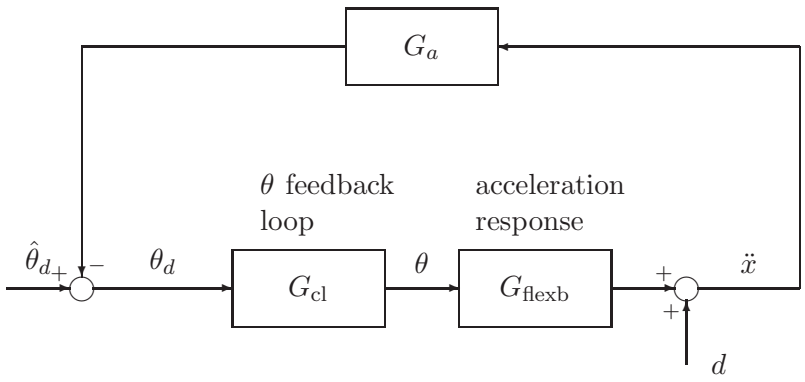

Figure 62: Simplified block diagram with disturbance input.

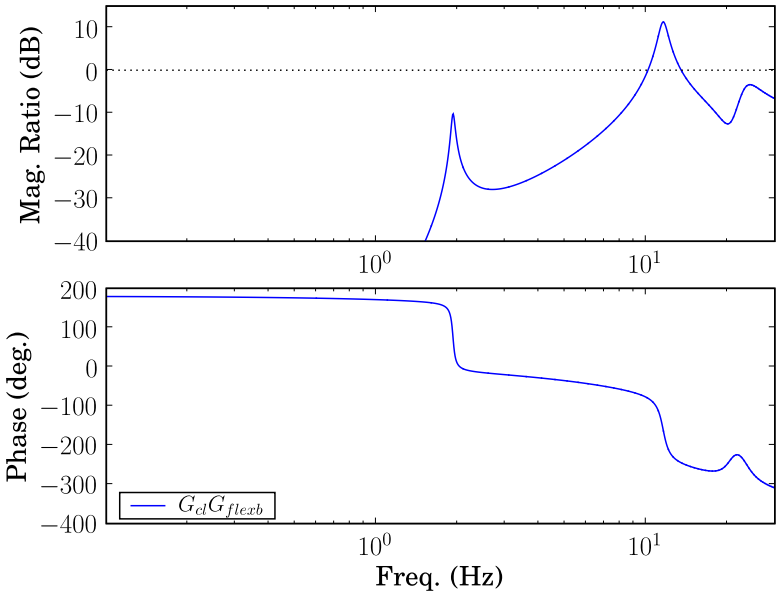

Figure 63: Bode plot of $G_{\text{cl}}G_{\text{flexb}} = \ddot{x}/\theta_d$.

transfer function between $\ddot{x}$ and d is

$$\frac{\ddot{x}}{d} = \frac{1}{1 + G_a G_{\text{cl}} G_{\text{flexb}}} \tag{179}$$

The goal of vibration suppression control design is to maximize $G_a G_{\text{cl}} G_{\text{flexb}}$ especially near the first natural frequency of the system so that the disturbance is rejected (i.e. the response of $\ddot{x}$ to d is minimized). Since G_{cl} is already specified by the θ feedback design and G_{flexb} is a property of the structure, the control design goal is really to maximize G_a near the first natural frequency of the system. While maximizing G_a, there are two constraints on the design. First, G_a must not drive the system unstable. Second, the low frequency amplitude of G_a must be constrained. Limit cycles have been experimentally observed when the low frequency amplitude of G_a is too high.

Figure 63 shows the Bode plot for $G_{\text{cl}}G_{\text{flexb}}$. Note that in order to meet the phase margin requirement the crossover frequency will need to be at a point where the phase is greater than –120°. This means that the crossover frequency must be to the left of the second natural frequency of the system (roughly 11.5 Hz). In order for this to happen G_a needs to suppress the second mode of the system.

5.2.3 Compensator Design

G_a needs to attenuate the accelerometer measurement for the second mode of the system in order for vibration suppression control to be effective. Two forms for G_a will be tried, a first-order compensator of the form

$$G_a = \frac{K_a p}{s+p} \tag{180}$$

and a second-order, low-pass Butterworth filter

$$G_a = \frac{K_a \omega_c^2}{s^2 + 2\zeta\omega_c s + \omega_c^2} \tag{181}$$

For both compensators, K_a is the low-frequency gain. p is the pole-location for the first-order compensator. ω_c is the corner-frequency of the Butterworth filter. Both p and ω_c are in rad/sec. $\zeta = 1/\sqrt{2}$ is the damping ratio of the Butterworth filter.

For both compensators, the control design problem consists of choosing values for the parameters of G_a so that the peak of the first mode of $G_a G_{\text{cl}} G_{\text{flexb}}$ is maximized while maintaining the required minimum phase margin and not exceeding $K_a = 20$.

The cost function for designing G_a is

```
1  def _fbcost(Ga):
2      ind1=thresh(f,1.25)
3      ind2=thresh(f,2.5)
4      accelB=CalcCompensatedAccelBode(Ga, Gcl ,Gfb)
5      pm=accelB.PhaseMargin(f)
6      peak1=max(accelB.dBmag()[ind1:ind2])
7      cost=100-peak1
8      Ka=Ga(0)
9      if pm<60.0:
10         cost+=(60.0-pm)**2
11     if Ka>20.0:
12         cost+=300*Ka
13     return cost
```

The input `Ga` is a compensator that has a numerator and denominator that are polynomials in s. Lines 2 and 3 determine the indices of the frequency vector that define the range around the first mode where $G_a G_{\text{cl}} G_{\text{flexb}}$ will be maximized. Line 4 calculates the loop transfer function $G_a G_{\text{cl}} G_{\text{flexb}}$. Line 5 calculates the phase margin. Line 6 determines the

maximum of the loop transfer function in the frequency range around the first mode. Line 7 specifies the initial costs as 100 minus the peak value of the loop transfer function. Line 8 determines the low-frequency gain. Lines 9 and 10 apply a penalty if the phase margin is too small. Lines 11 and 12 apply another penalty if the low-frequency gain is too large.

The θ feedback loop is already designed with $K_\theta = 1.684$. The optimized first-order design is $K_a = 20.00$ and $p = 2.07$ rad/sec (0.33 Hz). The optimized second-order design is $K_a = 20.00$ and $\omega_c = 8.48$ rad/sec (1.35 Hz).

Figures 65–66 compare Bode plots for optimized first and second-order compensators. Figure 64 compares the closed-loop actuator responses $\theta/\hat{\theta}_d$. Figures 64 and 65 show that the first-order compensator actually makes the system response at the second mode considerably worse. The vibration suppression controller with the first-order compensator adds damping to the first mode while removing damping from the second mode. The second-order compensator leads to better suppression of the first mode than the first-order compensator, as seen in Figures 65 and 66. Figure 65 shows that for the same θ motion, the system with the second-order compensator excites the first mode less. Figure 66 shows that the system with the second-order compensator rejects disturbances more. The second-order compensator clearly leads to a better design.

5.2.4 Simultaneous Design

This section compares sequential and simultaneous designs for G_θ and G_a. The sequential design scenario is essentially what has been described in sections 5.2.1 and 5.2.3, where G_θ is designed first and the result of $K_\theta = 1.684$ is used as an input to the G_a design. For both the sequential and simultaneous designs, G_θ is a proportional controller and G_a is a second-order compensator of the form of equation (181). For the simultaneous design, the cost function essentially takes the cost functions for the actuator and flexible base described previously adds them together:

```
1  def combinedcost(x):
2      global Gcl
3      Kth = x[0]
4      Gth = controls.Compensator(Kth,1)
5      Gcl = Gth(s)*Gp/(1.0+Gth(s)*Gp)
```

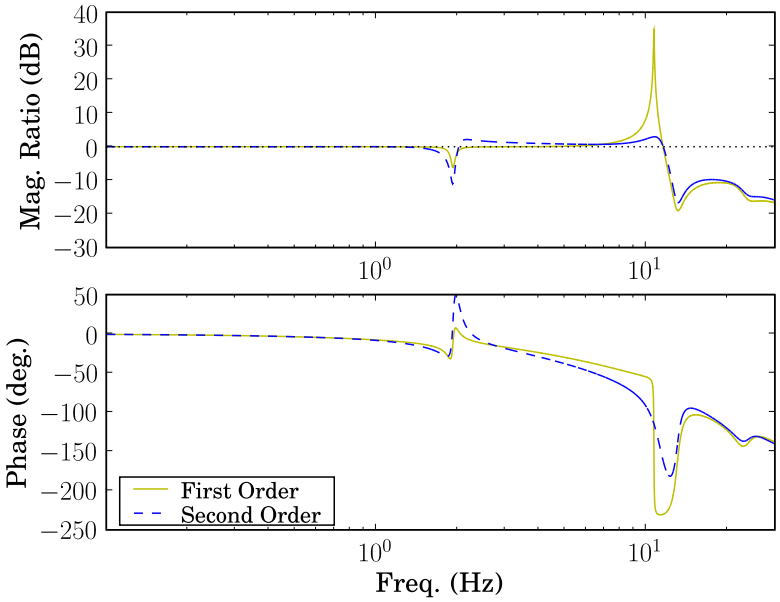

Figure 64: Closed-loop actuator Bode plots $\theta/\hat{\theta}_d$ for optimized first and second-order compensators.

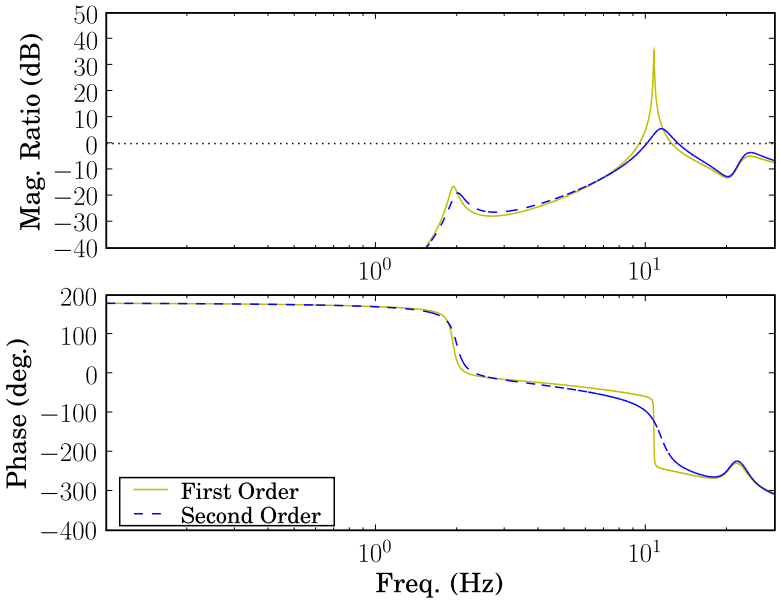

Figure 65: Closed-loop flexible base Bode plots $\ddot{x}/\hat{\theta}_d$ for optimized first and second-order compensators.

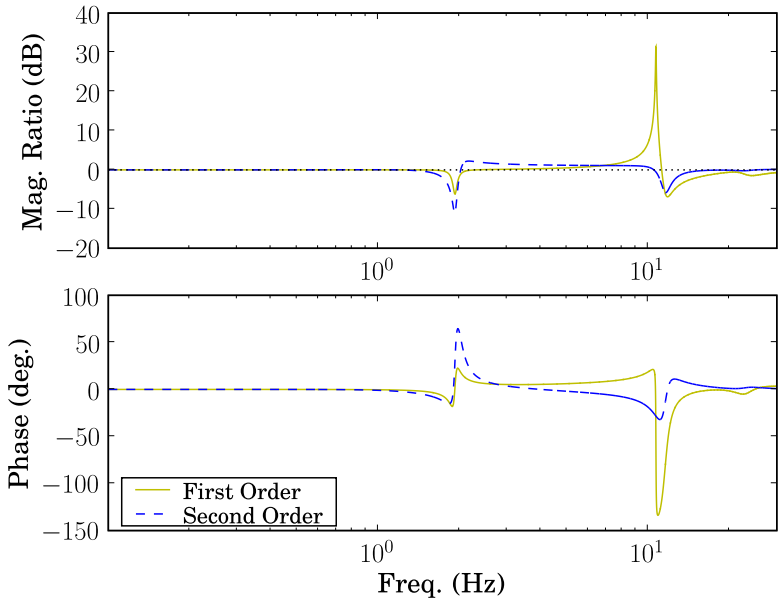

Figure 66: Closed-loop disturbance rejection Bode plots $\ddot{x}/d$ for optimized first and second-order compensators.

```
6       Ka = x[1]
7       fc = x[2]
8       Ga = controls.ButterworthFilterHzSO(fc, Ka)
9       actcost = _actcost(Gth)
10      fbcost = _fbcost(Ga)
11      return actcost+fbcost
```

The input x is a vector made up of the unknown coefficients [Kth, Ka, fc]. Line 4 creates a compensator that is a proportional controller. Line 5 calculates the closed-loop transfer function G_{cl} for the θ feedback loop. Line 8 creates the G_a compensator which is a second-order low-pass filter. Line 9 calculates the cost from the θ feedback loop. Line 10 calculates the cost from the $\ddot{x}$ feedback loop. Line 11 returns the combined cost.

Figures 67–69 show Bode plots comparing these two designs. Figure 67 shows that the simultaneous design has a lower cross-over frequency than the sequential design. Figures 68 and 69 show that this reduction in cross-over frequency is traded off against better vibration suppression. Figure 68 shows that both the first and second modes are excited less by actuator motion with the simultaneous design ($\ddot{x}/\hat{\theta}_d$ is reduced). Figure 69 shows that the simultaneous design is better able to reject disturbances at the first natural frequency. The

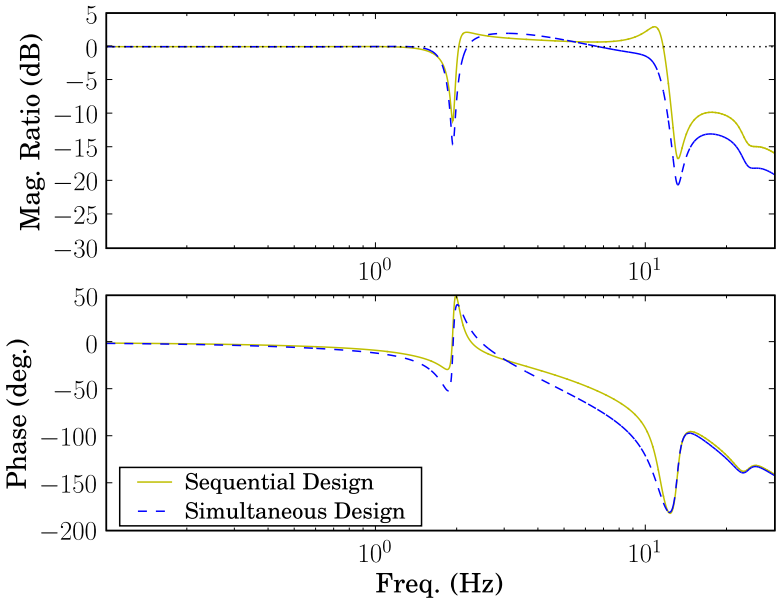

Figure 67: Closed-loop actuator Bode plots $\theta/\hat{\theta}_d$ for sequential vs. simultaneous design of G_θ and G_a.

simultaneous approach leads to a better design. Basically, a slight reduction in the actuator cross-over frequency leads to much less excitation of the second flexible mode which in turn makes more suppression of the first flexible mode possible.

5.2.5 Time-Domain Response

Figures 70 and 71 demonstrate the effectiveness of the vibration suppression design by comparing step responses with and without acceleration feedback. The input to the system is a unit step change (1°) in desired angle (θ_d for the system without $\ddot{x}$ feedback and $\hat{\theta}_d$ for the system with $\ddot{x}$ feedback). The output shown is the time response for acceleration of SAMII's base $\ddot{x}$. Figure 70 shows large amplitude acceleration from higher modes in the initial response. Figure 71 zooms in on the y-axis of Figure 70 to show the effectiveness of $\ddot{x}$ feedback at damping out vibration of the first mode.

Figure 72 zooms in on the x-axis of Figure 70, showing that the vibration suppression controller damps out both the first and second modes of vibration during the step response.

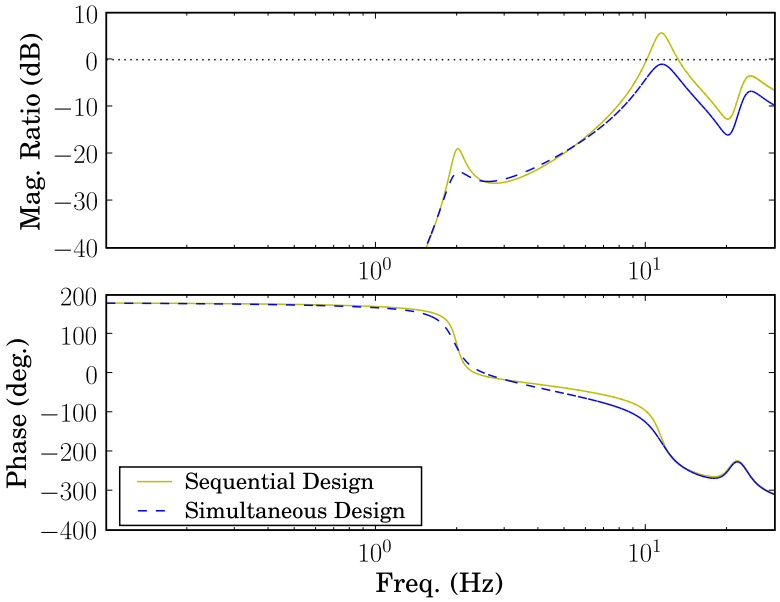

Figure 68: Closed-loop flexible base Bode plots $\ddot{x}/\hat{\theta}_d$ for sequential vs. simultaneous design of G_θ and G_a.

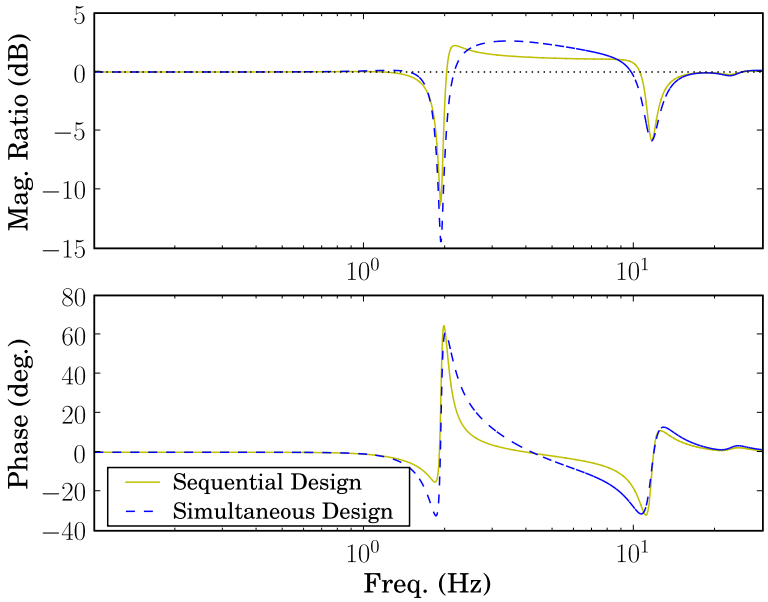

Figure 69: Disturbance rejection Bode plots $\ddot{x}/d$ for sequential vs. simultaneous design of G_θ and G_a.

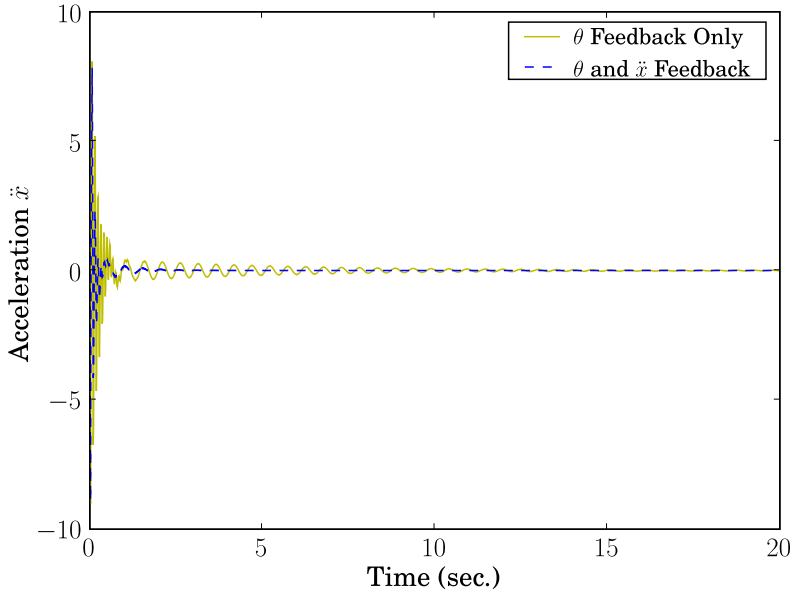

Figure 70: Base acceleration $\ddot{x}$ response to a step in desired angle with and without acceleration feedback.

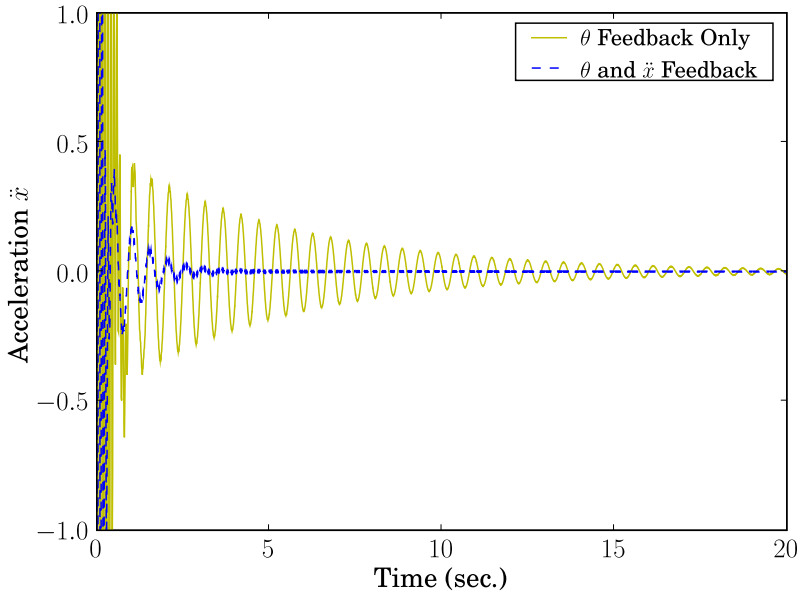

Figure 71: Base acceleration $\ddot{x}$ response to a step in desired angle with and without acceleration feedback (zooming in on the y-axis of Figure 70).

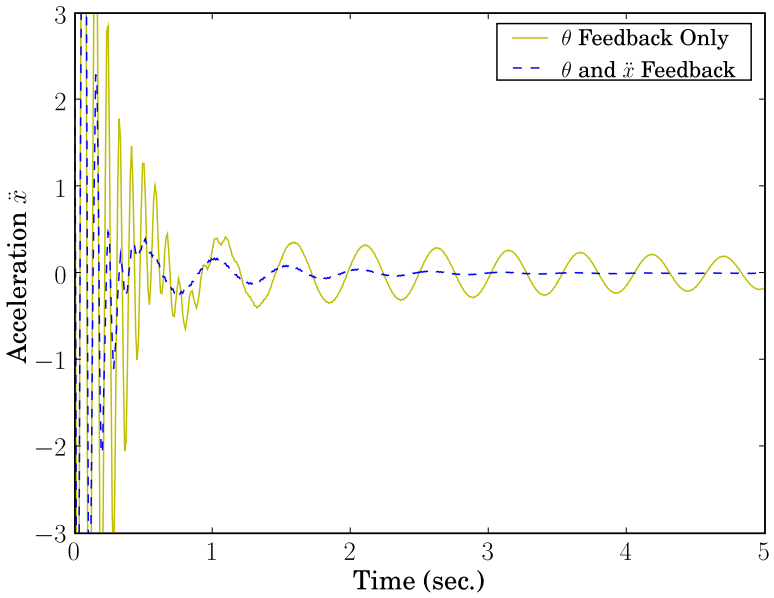

Figure 72: Base acceleration $\ddot{x}$ response to a step in desired angle with and without acceleration feedback (zooming in on the x-axis of Figure 70).

5.3 Direct Control Design

This section presents a control design technique that seeks to do pole-placement while avoiding modal discretization and the associated iteration. Any controller that can be expressed as a function of s can be plugged directly into the TMM model. Optimization algorithms could then be written that vary the control gains until the closed-loop system poles are driven to the desired locations. This approach avoids the iteration of the method of Book and Majette [9, 41] by not discretizing. The trade off is that it introduces the complication of searching for the roots of transcendental equations.

Carrying out this approach using a purely numerical implementation of the TMM leads to substantial convergence problems and very long execution times. A purely numerical implementation of the TMM involves substituting values for all element parameters and s, so that each element returns a numeric transfer matrix for a given value of s. These element matrices are multiplied together to form a system transfer matrix. Multiplying many of these element matrices involves a substantial number of floating-point operations which are time consuming and create opportunities for floating-point errors.

Implementing the TMM symbolically is a key to enabling this design approach. It leads to closed-form expressions for the closed-loop system response. This will involve infinite dimensional transfer functions for systems with continuous elements. For simple systems, inspecting this closed-loop expression may lead to insight into how to design the controller. For more complicated systems, it will be difficult to learn anything by direct inspection of the transfer function. Instead the transfer function will be used to generate Bode plots and in various numerical search algorithms to find pole-locations. Using this closed-form symbolic expression in optimization and numerical search routines will be much faster, will converge more readily, and will avoid floating-points problems compared to doing the same analysis using a purely numeric approach.

5.3.1 Computer Algorithm and Usage

Symbolic implementation of the TMM does not mean simply typing a bunch of matrices into a computer algebra system by hand and operating on them. That would be error prone and time consuming. A software package for TMM analysis has been created as part of this work. The user interacts with the this software like a higher level language for describing systems using the TMM. The software is written as a module in the Python programming language. Python is a dynamic object-oriented programming language well-suited to rapid software development.

Symbolic capabilities have been built into the Python TMM analysis software. When symbolic analysis is requested, Python constructs an input script for Maxima, the symbolic engine used in this work. Maxima was selected for several reasons, but primarily because Python can call it in the background, pass it an input script, and handle its output in a way that the user never has to leave Python nor learn Maxima. Maxima is also cross-platform, open-source, and free.

Once the Maxima input script is generated, Python calls Maxima passing this input script. The script commands Maxima to output its results to FORTRAN files as well as a LaTeX file. The LaTeX file allows the user to inspect the Maxima output in an easy-to-read format. The FORTRAN files are handled by Python in two ways. The files are converted

from FORTRAN syntax to Python and these files can be used directly as Python modules or edited if needed. Python also adds header information to the FORTRAN files and compiles them. The user then has access to Python modules that are really compiled FORTRAN code. These modules run faster than pure Python modules for some numerically intense applications. This can be beneficial for some optimization routines where the Bode functions will be called many times. Provided that good header files already exist, the user can take advantage of the speed of FORTRAN running within Python without having to write any actual FORTRAN code themselves. In summary, symbolic analysis is handled completely by Python and is transparent to the user. It also leads to faster and more numerically stable execution than a purely numeric TMM implementation.

5.3.2 Analysis Example

Symbolic analysis for control design will most often take one of two forms. The first is finding a symbolic expression for the characteristic determinant of the system. The second is determining the closed-loop transfer function with an augmented input modeling actuation of some kind. The analysis for actuated systems consists of five main steps:

1. Find symbolic transfer matrices for each element
2. Multiply the element matrices to find the system transfer matrix
3. Apply the boundary conditions to determine the state vector at the base as a function of s
4. Use the element transfer matrices and the state vector at the base to find the state vector at the output locations
5. Post-process the output state-vectors if needed (to find velocities or accelerations or relative signals)

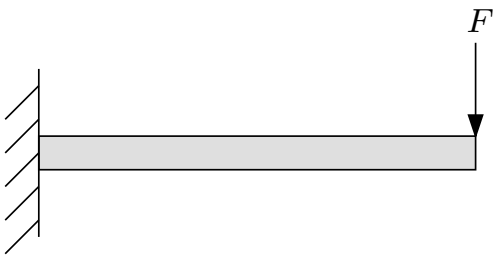

Figure 73: Sketch of a cantilever beam with a force at its free end

As a simple example, consider a cantilever beam driven by a force at the free end as shown in Figure 73. If $\mathbf{U}_{\text{beam}}$ is the transfer matrix of a beam element and

$$\mathbf{U}_{\text{F}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & F \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{182}$$

is the transfer matrix for the forcing shown in Figure 73, then the system model will be

$$\mathbf{z}_{\text{end}} = \mathbf{U}_{\text{F}}\ \mathbf{U}_{\text{beam}}\ \mathbf{z}_{\text{base}} \tag{183}$$

where

$$\mathbf{z} = \begin{Bmatrix} w \\ \theta \\ M \\ V \\ 1 \end{Bmatrix} \tag{184}$$

The boundary conditions require that displacement and rotation are zero at the base, and that force and moment are zero at the tip. (The tip for this system is after the forcing matrix. That may seem slightly odd conceptually, but it is easier to write algorithms where boundary conditions are always zero and forcing is always coming from forcing elements rather than treating $F(s)$ as a boundary condition.) Plugging the boundary conditions into

the system model produces

$$\begin{Bmatrix} w_{\text{end}} \\ \theta_{\text{end}} \\ 0 \\ 0 \\ 1 \end{Bmatrix} = \mathbf{U}_{\text{sys}} \begin{Bmatrix} 0 \\ 0 \\ M_{\text{base}} \\ V_{\text{base}} \\ 1 \end{Bmatrix} \tag{185}$$

where

$$\mathbf{U}_{\text{sys}} = \mathbf{U}_{\text{F}}\ \mathbf{U}_{\text{beam}} \tag{186}$$

The third and fourth row of equation (185) are

$$\begin{Bmatrix} 0 \\ 0 \end{Bmatrix} = \begin{bmatrix} \mathbf{U}_{\text{sys33}} & \mathbf{U}_{\text{sys34}} \\ \mathbf{U}_{\text{sys43}} & \mathbf{U}_{\text{sys44}} \end{bmatrix} \begin{Bmatrix} M_{\text{base}} \\ V_{\text{base}} \end{Bmatrix} + \begin{Bmatrix} \mathbf{U}_{\text{sys35}} \\ \mathbf{U}_{\text{sys45}} \end{Bmatrix} \tag{187}$$

which can be solved for the unknown elements of the base vector:

$$\begin{Bmatrix} M_{\text{base}} \\ V_{\text{base}} \end{Bmatrix}_{\text{base}} = - \begin{bmatrix} \mathbf{U}_{\text{sys33}} & \mathbf{U}_{\text{sys34}} \\ \mathbf{U}_{\text{sys43}} & \mathbf{U}_{\text{sys44}} \end{bmatrix}^{-1} \begin{Bmatrix} \mathbf{U}_{\text{sys35}} \\ \mathbf{U}_{\text{sys45}} \end{Bmatrix} \tag{188}$$

Once M_{base} and V_{base} are known, $\mathbf{z}_{\text{base}}$ is known and the state vector at the end of the beam can be found from

$$\mathbf{z}_{\text{beam}} = \mathbf{U}_{\text{beam}}\ \mathbf{z}_{\text{base}} \tag{189}$$

and if the desired output is simply the displacement at the beam tip, then the transfer function between F and w_{beam} will simply be the first element of $\mathbf{z}_{\text{beam}}$. All of this analysis can be carried out symbolically if expressions for the element transfer matrices exist. This analysis process is automated by the software and transparent to the user. The resulting transfer function for this simple system is

$$\frac{w_{\text{beam}}}{F} = -\frac{2\ (\cos\beta\ \sinh\beta - \cosh\beta\ \sin\beta)\ \ L^3}{\beta^3\ \left(\sinh^2\beta - \sin^2\beta - \cosh^2\beta - 2\ \cos\beta\ \cosh\beta - \cos^2\beta\right)\ EI} \tag{190}$$

5.3.3 Verification

As a verification of this symbolic TMM analysis, consider solving the same problem analytically. A general solution for the spatial differential equation for a beam in bending is

$$y(x) = c_3 \sinh\left(\frac{\beta\, x}{L}\right) + c_1 \sin\left(\frac{\beta\, x}{L}\right) + c_4 \cosh\left(\frac{\beta\, x}{L}\right) + c_2 \cos\left(\frac{\beta\, x}{L}\right) \tag{191}$$

For a cantilever beam forced at the free end the boundary conditions are

$$y(0) = c_4 + c_2 = 0 \tag{192}$$

$$\left.\frac{\partial y}{\partial x}\right|_{x=0} = \frac{\beta\ (c_3 + c_1)}{L} = 0 \tag{193}$$

$$M(L) = \frac{\beta^2\ (\cosh\beta\, c_4 + \sinh\beta\, c_3 - \cos\beta\, c_2 - \sin\beta\, c_1)\ EI}{L^2} = 0 \tag{194}$$

$$V(L) = -\frac{\beta^3\ (\sinh\beta\, c_4 + \cosh\beta\, c_3 + \sin\beta\, c_2 - \cos\beta\, c_1)\ EI}{L^3} = -F \tag{195}$$

Solving equations (192)–(195) for c_1, c_2, c_3, and c_4 and substituting the results into equation (191) and then evaluating at $x = L$ produces the transfer function from force at the tip to displacement at the tip:

$$\frac{w_{\text{beam}}}{F} = -\frac{2\ (\cos\beta\, \sinh\beta - \cosh\beta\, \sin\beta)\ L^3}{\beta^3\ \left(\sinh^2\beta - \sin^2\beta - \cosh^2\beta - 2\, \cos\beta\, \cosh\beta - \cos^2\beta\right)\ EI} \tag{196}$$

which is identical to equation (190).

5.3.4 Symbolic Control Design

Obtaining a symbolic TMM model of a system, like that of equation (190), is the first step in the direct TMM control design process. The remainder of the process will be introduced through another example. Book and Majette [9, 41] applied their control design technique to a robot made up of two flexible links connected by an actuated joint and clamped at its base. This system is shown in Figure 74. The torsional spring K_p and damper K_d represent collocated PD control of the joint. Control design using the direct method will be compared to the method of Book and Majette for this system.

A transfer matrix model of this system is

$$\mathbf{z}_{\text{tip}} = \mathbf{U}_{\text{beam2}}\mathbf{U}_{\text{sd}}\mathbf{U}_{\text{beam1}}\, \mathbf{z}_{\text{base}} \tag{197}$$

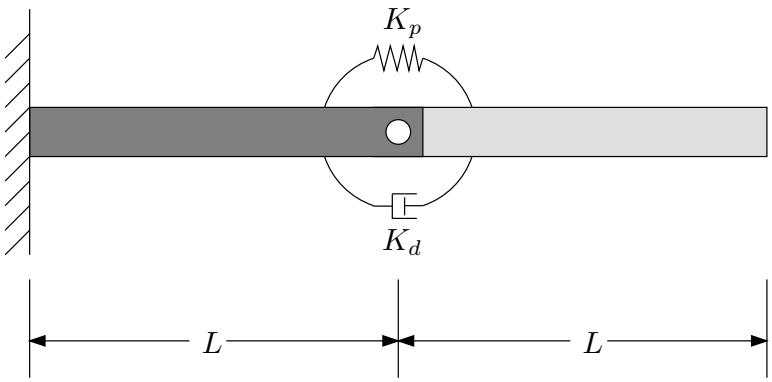

Figure 74: The two-flexible-link robot used by Book and Majette [9, 41] for TMM pole-placement control design.

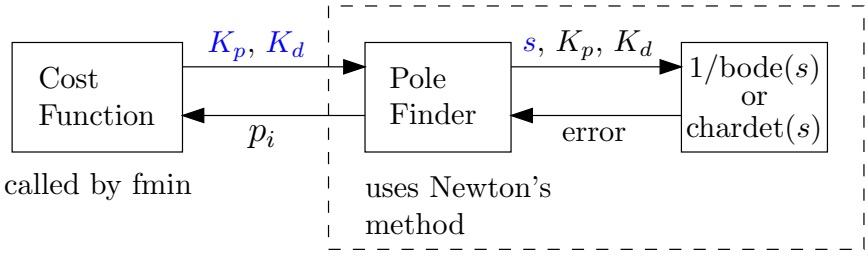

Figure 75: Symbolic control design algorithm.

where

$$\mathbf{U}_{\text{sd}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & \dfrac{1}{K_d s + K_p} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{z} = \begin{Bmatrix} w \\ \theta \\ M \\ V \end{Bmatrix} \tag{198}$$

$\mathbf{U}_{\text{sd}}$ is a torsional spring/damper transfer matrix and $\mathbf{U}_{\text{beam1}}$ and $\mathbf{U}_{\text{beam2}}$ are transfer matrices for beam elements. For this problem, the beams are identical and their parameters are $\mu = 1$, $EI = 1$, and $L = 0.5$. The control design problem is to find values for K_p and K_d so that the dominant closed-loop poles are placed at $s = -0.3 \pm 3j$.

5.3.4.1 Algorithm

The symbolic control design algorithm is depicted in Figure 75. At the top level (on the left of the diagram), there is a cost function that defines an error to be minimized. This cost is based on the difference between the current pole locations p_i and the desired pole locations. The optimization routine seeks to minimize this cost by varying the control gains.

In order to calculate this cost, the poles must be found. This is the responsibility of the pole-finding function (center box in Figure 75). The pole-finding function seeks roots of the characteristic equation which are the pole locations. The control gains are constant inputs to the pole-finding function. It outputs the values of s that are the current pole locations. These poles are found by numerically searching for the roots of the characteristic equation. This numerical search requires good initial guesses. The poles from the previous step are used as initial guesses.

The symbolic TMM model is generated by the following Python code:

```
1  #create TMMElements
2  myspring = TorsionalSpringDamper({'k':[1],'c':[1]},maxsize=4,
       unknownparams=['k','c'])
3  bp = {'mu':1.0,'EI':1.0,'L':0.5}
4  mybeam = BeamElement(bp,maxsize=4,symname='Ubeam1',symlabel='bz1')
5  mybeam2 = BeamElement(bp,maxsize=4,symname='Ubeam2',symlabel='bz2')
6  #create TMMSystem
7  mysys = TMMSystem([mybeam,myspring,mybeam2],'free','fixed')
8  #generate symbolic functions
9  prefix = 'sym_majette_TMM'
10 mysys.SymCharDetAll(curvefit=True,sub=False)
11 #compile FORTRAN
12 mysys.CallF2py(['chardet_out.f'])
```

Line 2 creates the TorsionalSpringDamper element that models the actuator. Line 3 defines the beam parameters. Lines 4 and 5 create the beam elements. Line 7 creates the TMMSystem. Line 10 runs the symbolic analysis to find the characteristic determinant. Line 12 compiles the symbolic output into a FORTRAN module that Python can import.

The pole finding function is

```
1  def FindPole(xvect,poleguess):
2      curpole = newton(chardet,poleguess,args=(xvect,))
3      return curpole
```

where the inputs to this function are xvect = [Kp, Kd] and the initial guess to use for Newton's method poleguess. Line 2 finds the value of s that corresponds to a system pole while keeping Kp and Kd constant.

The cost function is

```
1  def MyCost(xvect,despole,poleguess=None):
```

Table 3: Results from symbolic TMM control design.

actual pole location	-0.300000+3.000000j
K_p	0.94521384
K_d	0.22834017
abs(error)	-0.000000+0.000000j
execution time	0.08 seconds

```
2       if poleguess is None:
3           poleguess = despole
4       curpole = FindPole(xvect,poleguess)
5       myerror = despole-curpole
6       return abs(myerror)
```

Line 4 calls the above FindPole function to find the current pole location. Lines 5 and 6 calculate the cost to be the error between the actual and desired pole locations.

The control design can be run by executing

```
despole = -0.3+3.0j
Kp, Kd = fmin(MyCost,[0.01,0.0],args=(despole,),ftol=1e--9)
```

where [0.01, 0.0] are the initial guesses for [K_p, K_d]. The call to fmin takes approximately 0.08 seconds. The results are shown in Table 3

5.3.5 Comparison to the Method of Book and Majette

The approach of section 5.3.4 will now be compared to the method of Book and Majette for the same control design problem. The method of Book and Majette is based on converting the TMM model to a state-space model through modal discretization and then using a pole-placement technique to find the desired gains for the state-space model. If the controller significantly affects the mode shapes and natural frequencies, iteration will be required. The state-space model will be

$$\begin{aligned} \frac{d\mathbf{x}}{dt} &= \mathbf{\Lambda x} + \mathbf{Bu} \\ \mathbf{y} &= \mathbf{Cx} \end{aligned} \tag{199}$$

where $\mathbf{\Lambda}$ has on its diagonal the eigenvalues of the modes used in the discretization. $\mathbf{C}$ can be determined from the system mode shapes. The $\mathbf{B}$ matrix is yet to be determined. The

matrix transfer function for the system is

$$\mathbf{G}(s) = \mathbf{C}(s\mathbf{I} - \mathbf{\Lambda})^{-1}\mathbf{B} \tag{200}$$

Numerical values for the matrix transfer function can also be found from a TMM Bode response at various values of s. $\mathbf{B}$ can then be found by a least-squares curve fit of the equation

$$\mathbf{G}_{\text{TMM}} \approx \mathbf{G}(s) = \mathbf{C}(s\mathbf{I} - \mathbf{\Lambda})^{-1}\mathbf{B} \tag{201}$$

The need for iteration can be understood by thinking about the system of Figure 74. For very small values of K_p and K_d, this system will have a first natural frequency of nearly zero and the mode will essentially consist of two rigid links with the second pivoting about the joint. As K_p grows very large, the system will approach a cantilevered beam of length $2L$. The control gains will affect not only the natural frequencies, but also the mode shapes of the system. This will affect the modal discretization used to generate the state-space model. Note that any approach to control design that depends on a modal representation of the system will have this same need for iteration.

Applying the method of Book and Majette to the system in Figure 74 takes 3.9 seconds and leads to the results shown in Table 4. Note that this method converges to the same solution as the symbolic TMM approach (Table 3), so that the methods validate one another. It should also be pointed out that the symbolic approach is easier to implement because it avoids the modal discretization and it runs much faster (approximately 40 times as fast for this problem).

5.3.6 Pole-Placement Design for SAMII

As a next step in control design complexity, consider simultaneous design of the position feedback and vibration suppression control for SAMII, as shown in Figure 76. The control scheme uses proportional control for θ and proportional control with a low-pass filter for $\ddot{x}$. This is the same form for the control as used in the Bode optimization of section 5.2. A controller of this form is known to work fairly well based on experimental results.

The dynamics of this system are dominated by one real pole and three pairs of complex poles. The system has an infinite number of poles and the dynamics of any could become

Table 4: Summary of the iteration process for the state-space design methodology of Book and Majette for the system in Figure 74.

Iteration	K_p	K_d	Eigenvalue	Error (abs)
0	0.01000000	0.00000000	-0.000000+0.485676j	2.532158
1	0.38861147	0.02565663	-0.103581+2.340975j	0.687673
2	0.70769571	0.07418619	-0.159152+2.733151j	0.301739
3	0.90009130	0.12972375	-0.200068+2.888061j	0.150055
4	0.98011847	0.17500927	-0.232876+2.957729j	0.079325
5	0.99570142	0.20407931	-0.258310+2.989046j	0.043105
6	0.98606356	0.21957749	-0.276411+3.001490j	0.023636
7	0.97177247	0.22654284	-0.288012+3.004937j	0.012965
8	0.96026864	0.22905671	-0.294675+3.004685j	0.007093
9	0.95280668	0.22959107	-0.298086+3.003363j	0.003870
10	0.94857000	0.22940842	-0.299611+3.002071j	0.002107
11	0.94642341	0.22906731	-0.300167+3.001134j	0.001146
12	0.94546725	0.22877188	-0.300286+3.000553j	0.000623
13	0.94511758	0.22856918	-0.300245+3.000234j	0.000338
14	0.94504021	0.22844810	-0.300167+3.000077j	0.000184
15	0.94506385	0.22838349	-0.300099+3.000010j	0.000100
16	0.94511097	0.22835283	-0.300053+2.999988j	0.000054
17	0.94515241	0.22834040	-0.300025+2.999984j	0.000029
18	0.94518107	0.22833667	-0.300010+2.999988j	0.000016
19	0.94519830	0.22833651	-0.300003+2.999992j	0.000009

significant if a pole was driven unstable. In practice however, the locations of these poles are sufficient to characterize the response of the system. The real pole comes from the position feedback loop. Two of the complex poles are the first two flexible modes of the system. The other complex pole comes from the low-pass filter in the acceleration feedback loop. The control design goal is to find values for K_θ, K_a, and ω_c that place the real pole and the complex poles from the first mode at the desired locations while ensuring that the mode 2 and filter poles stay to the left of prescribed boundaries. The pole-placement problem is illustrated in Figure 77.

The cost function for this design is

```
1 def mycost(xvect,mod1=afbvarK0,mod2=afbvarK1):
2     global guesses
3     global prevx
4     Kth=xvect[0]
5     Ka=xvect[1]
6     wc=xvect[2]
```

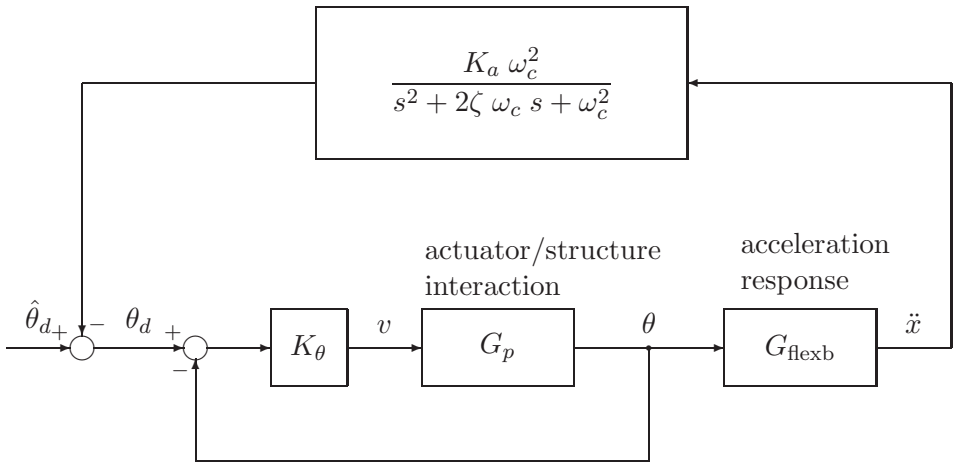

Figure 76: Block diagram of the pole-placement control problem for SAMII.

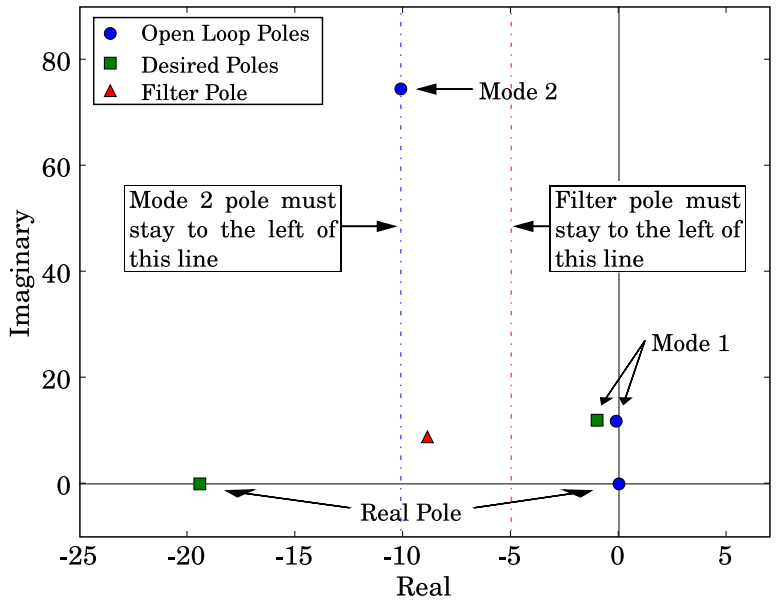

Figure 77: Open-loop and desired closed-loop pole-locations for the SAMII pole-placement design problem.

```
7     #search for poles, call adaptive interpolation if poles found are
          too far away from the initial guesses
8     try:
9         pr, p1, pf, p2 = afbpoles(Kth,Ka,wc,guesses,mod1,mod2)
10    except pException:
11        if prevx is not None:
12            pr, p1, pf, p2 = adaptinterp(prevx, xvect, guesses)
13        else:
14            raise
15    guesses=[pr,p1,pf,p2]#use previous solution for next initial guess
16    prevx=xvect
17    #set weights for real pole and mode 1 pole
18    wtr=1.0
19    wt1=100.0
20    costout=wtr*(dpr1-pr)**2+wt1*(abs(p1-dpf1))**2
21    #add penalties to the cost if the mode 2 or filter poles are to the
          right of their thresholds
22    if real(p2)>real(boundary2):
23        penalty=(1.0+real(p2)-real(boundary2))**3
24        costout+=penalty
25    if real(pf)>real(boundaryf):
26        penalty=(1.0+real(pf)-real(boundaryf))**3
27        costout+=penalty
28    return abs(costout)
```

The inputs to the cost function are xvect=[Kth, Ka, wc] (the vector of unknown coefficients) and mod1 and mod2 which are FORTRAN modules for the closed-loop transfer functions $\theta/\hat{\theta}_d$ and $\ddot{x}/\hat{\theta}_d$. Lines 8 and 9 try to find the four closed-loop poles of interest. pr is the real pole. p1 and p2 are the first and second flexible poles. pf is the filter pole. If the function afbpoles does not converge to correct values for the poles, it raises the pException. Line 10 catches this exception. As long as a valid previous solution exists, line 12 will initiate the adaptive interpolation algorithm adaptinterp (this algorithm will be discussed in section 5.3.6.1). If this fails, the exception will be re-raised in line 14 and the algorithm will exit with an error. If either line 9 or line 12 converge to valid pole locations, the algorithm will continue and lines 15 and 16 will save the results as global variables to be used as initial guesses next time. Lines 18 and 19 define the weights for the real pole (wtr) and the pole from mode 1 (wt1). Note that the weight for the mode 1 pole is set considerably

higher than that for the real pole (100:1). While the desired mode 1 pole location is not far from the open-loop location, this small movement to the left leads to significant vibration reduction. If the weights for the two poles are the same, the initial error caused by the real pole is much larger than that from the mode 1 pole and the algorithm would focus on reducing the error from the real pole. Line 20 defines the cost before any penalties are added. The cost is the squared difference between the desired and actual pole locations times the respective weights. Lines 22–27 apply penalties if the mode 2 or filter poles are to the right of their boundaries. The penalties are continuous and get higher as the actual pole location moves to the right of its boundary. This ensures that the cost function correctly guides optimization algorithms that use numerical derivatives or similar approaches in their search.

5.3.6.1 Pole-Tracking Algorithm

The most challenging portion of the pole-placement algorithm is tracking all four of the dominant poles as the optimization algorithm is varying the parameters. This control design approach hinges on being able to find all four of these poles and tell them apart from one another. This is complicated by the fact that the pole-finding algorithm can be sensitive to initial guesses. This problem is much more complicated numerically than the control design for the system of Figure 74.

It would seem fairly straight forward to search for the roots in the neighborhood of the desired pole locations. Initial implementations of this approach kept running into problems as the optimization tried unanticipated combinations of gains that placed the poles outside the expected regions. The numerical domains of attraction of the poles also grew very small with certain gain combinations, so that the algorithm could not be made to converge to the correct pole. There were also problems separating the mode 1 pole from the filter pole for certain values of ω_c.

A robust algorithm for pole-tracking has been created. The approach requires good initial guesses for each of the four poles for the starting gain values used in the optimization, i.e. for the values of K_θ, K_a, and ω_c that will be used as starting points for the optimization

algorithm, good initial guess for the pole-locations are also required. This may require some work on the part of the user to initialize the algorithm with good guesses, possibly by doing a brute-force search over a fine grid in the parameter space. This will only be required once.

The pole-tracking portion of the algorithm tells the poles apart based on the initial guesses used to find each pole. If the user wants to track 4 poles, he or she must provide 4 initial guesses to the pole-finding function. For each pole, the pole finder will check to see that the root it finds is not very fair from the initial guess. The poles are found using Newton's method. For example, if the pole the algorithm thinks is from mode 2 is not far away from the initial guess for the mode 2 pole, then it must really be the mode 2 pole. If the pole found based on the mode 2 initial guess is far away from the initial guess, the algorithm may have converged to a wrong solution. Each time the algorithm correctly finds the poles, it saves the current poles to be used as initial guesses in the next step.

Keep in mind that this section is talking about the pole-finding portion of the algorithm. This pole finder is locating the poles in order to pass them to the top level function that will use the current pole locations to calculate the current cost. To say that the pole-finding function converged means only that the current pole locations have been found, not that the optimal control gains have been found.

In the event that the algorithm fails to converge to poles that are near the corresponding initial guesses, an adaptive interpolation algorithm is used. This can happen if the optimization is trying to vary the gains by a considerable amount in between steps or if the current combinations of gains leads to a small numerical domain of attraction for a certain pole. This adaptive interpolation algorithm is completely automatic – it requires no input from the user and the user doesn't have to call it explicitly.

When the pole finder fails to find a pole, a line is calculated between the previous location in the parameter space (where the poles converged correctly and are known) and the current location where the poles cannot be correctly identified. The mid-point of this line is first tried. If the poles converge correctly at the mid-point, the poles from the mid-point are used as initial guesses to try at the desired end-point. If the mid-point fails to converge, the line segment is cut in half again. A maximum number of segments to be tried

is set up and if it is exceeded, the algorithm varies the parameters one at a time in a similar fashion that iteratively cuts the line between current and desired location in half until the pole converges correctly. All of this is done on a pole-by-pole basis so that computational resources are not wasted on poles that are converging correctly.

5.3.6.2 An Example

As an example of the adaptive interpolation algorithm, there is a time in the pole-placement design when the optimization algorithm is trying to move from K_θ=1.06138, K_a=23.35713, ω_c=15.23394 (call this point A) to K_θ=0.89655, K_a=18.27556, ω_c=12.66609 (call this point B). This example is illustrated in Figure 78. At point A, the filter pole is $p_{fA} = -11.55917 + 15.96061j$. If p_{fA} is used as the initial guess for point B, the pole-finding algorithm returns $-10.62022 + 72.05144j$, which is not a valid location for the filter pole (it is the mode 2 pole). The interpolation algorithm defines a new point C midway between A and B at K_θ=0.97896, K_a=20.81634, ω_c=13.95002. Using p_{fA} as an initial guess at point C, the pole-finding algorithm returns $-11.13439 + 71.55562j$, which is still the mode 2 pole. The algorithm then defines point D midway between A and C (K_θ=1.02017, K_a=22.08674, ω_c=14.59198). From initial guess p_{fA}, at point D the algorithm converges to $p_{fD} = -10.89708+14.84810j$. The algorithm then attempts to move from D to C. Using p_{fD} as the initial guess at point C, the algorithm converges to $p_{fC} = -10.24573+13.76899j$. The algorithm then defines a new point E midway between B and C. The algorithm converges from C to E ($p_{fE} = -9.61408 + 12.72120j$) and from E to B ($p_{fB} = -9.01302 + 11.70679j$).

The red dashed lines in Figure 78 represent failures to converge from A to B and from A to C. The blue solid lines represent successful steps from A to D, D to C, C to E, and E to B.

The results for the pole-placement design problem are shown in Figure 79. The real pole and mode 1 pole have been placed at the desired locations and the filter pole and mode 2 pole have stayed to the left of their prescribed boundaries. The algorithm returns K_θ=1.00001, K_a=18.00024, ω_c=12.56653 rad/sec or 2.0000 Hz. These values are experimentally known to be correct.

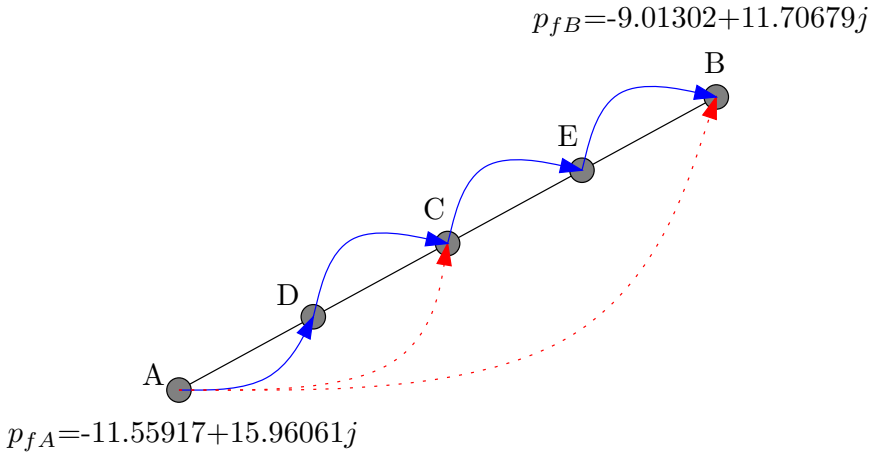

Figure 78: Adaptive interpolation algorithm moving from point A to point B.

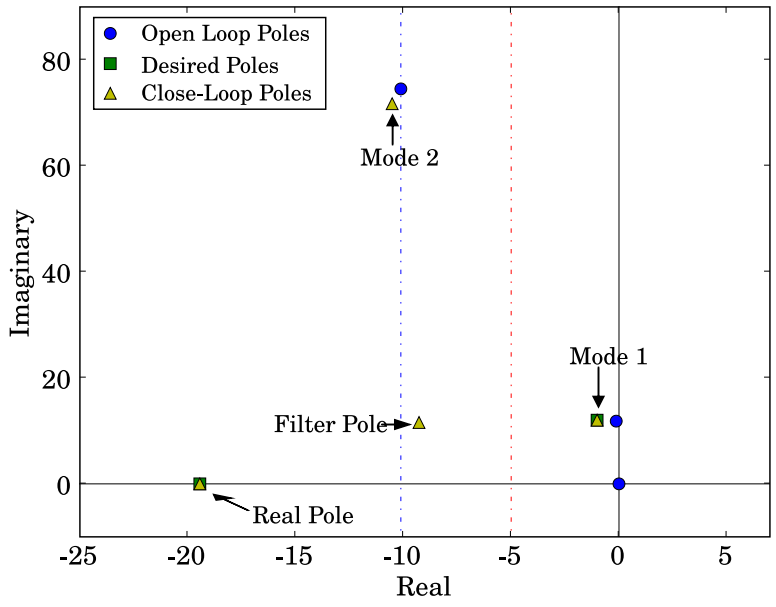

Figure 79: Results from the pole-placement control design.

An interesting software design note is that pole-convergence problems are handled by throwing and catching exceptions in Python. An exception is basically a specific error that can be user created. Functions can be told to look for specific errors that may be caused by certain portions of code, and contingencies can be set up to deal with the errors. Failing to converge or converging to incorrect poles throws an exception. If the exception is caught and the problem is over-come through some contingency, then the error is eliminated. Otherwise the exception is re-raised and execution stops. This makes it much easier to design code than having to anticipate and alleviate any possible source of these errors and building in ways for the code to exit rather than being stuck in some non-converging loop.

5.3.6.3 Limitations of the Pole-Tracking Algorithm

The pole-tracking algorithm will encounter problems for systems whose dynamics differ significantly from those of SAMII and this will require care on the part of the user. Systems whose poles cross as control gains are varied will be challenging and may require the development of a specific pole-crossing algorithm or the problem may need to be broken up into two separate problems before and after the crossing. Systems that start out with repeated poles, such as multiple poles at the origin will also require a careful and intelligent approach as well as the extension of the algorithm presented here.

5.3.7 SAMII Optimization

As an even more challenging and interesting control design problem, consider asking the algorithm to optimize SAMII's closed-loop pole locations rather than simply place them at some desired location. One possibility would be to drive the mode 1 pole and the real pole as far as possible to the left while still meeting the same requirements for the mode 2 and filter poles. The cost function would be

```
1 def mycost2(xvect,mod1=afbvarK0,mod2=afbvarK1):
2     global guesses
3     global prevx
4     Kth = xvect[0]
5     Ka = xvect[1]
6     wc = xvect[2]
7     try:
```

```
8           pr, p1, pf, p2 = afbpoles(Kth,Ka,wc,guesses,mod1,mod2)
9       except pException:
10          if prevx is not None:
11              pr, p1, pf, p2 = adaptinterp(prevx, xvect, guesses)
12          else:
13              raise
14      guesses = [pr, p1, pf, p2]#use previous solution for next initial
            guess
15      prevx = xvect
16      wtr = 1.0
17      wt1 = 10.0
18      w1, z1 = stowz(p1)
19      cost1 = 1.0/(1.0+w1*z1)
20      costr = 1.0/(1.0-pr)
21      costout = wtr*costr+wt1*cost1
22      if real(p2)>real(bpf2):
23          penalty = (1.0+real(p2)-real(bpf2))**3
24          costout += penalty
25      if real(pf) > real(bpfr):
26          penalty = (1.0+real(pf)-real(bpfr))**3
27          costout += penalty
28      if Ka>20.0:
29          costout += ((Ka-20)*100)**4
30      return abs(costout)
```

Lines 1-16 are identical to the cost function for pole-placement. Line 17 defines the weighting for the mode 1 pole. It is different from what was used in the pole-placement design because the cost is defined much differently. Line 18 calls a function that converts the complex value for p1 to its natural frequency and damping ratio w1 and z1. Line 19 defines the cost for the first flexible pole cost1. Line 20 defines the cost for the real pole costr. Note that these costs will grow smaller as the poles moves to the left. Line 21 uses the weighting factors to define the total cost. Lines 22–24 define a penalty if the second flexible pole moves to the right of its boundary. Lines 25–27 define a penalty if the filter pole moves to the right of its boundary. Lines 28 and 29 apply a penalty if K_a is too high.

Figures 80–82 compare the results of the pole-location-based optimization to those of the Bode-based optimization from section 5.2. Figure 80 shows the actuator Bode plot $\theta/\hat{\theta}_d$. Figure 81 shows the flexible base Bode plot $\ddot{x}/\hat{\theta}_d$. Figure 82 shows the disturbance rejection

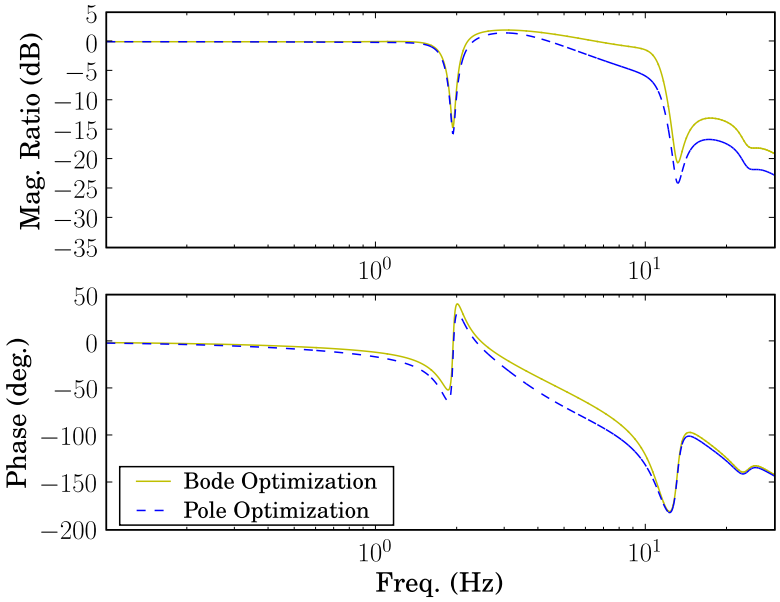

Figure 80: Comparison of actuator Bode plots $\theta/\hat{\theta}_d$ for the Bode and pole optimization control designs.

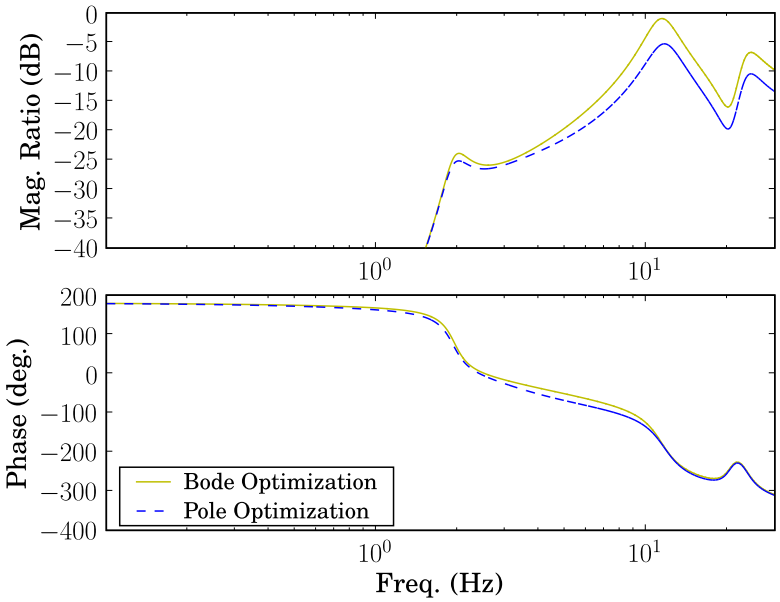

Figure 81: Comparison of flexible base Bode plots $\ddot{x}/\hat{\theta}_d$ for the Bode and pole optimization control designs.

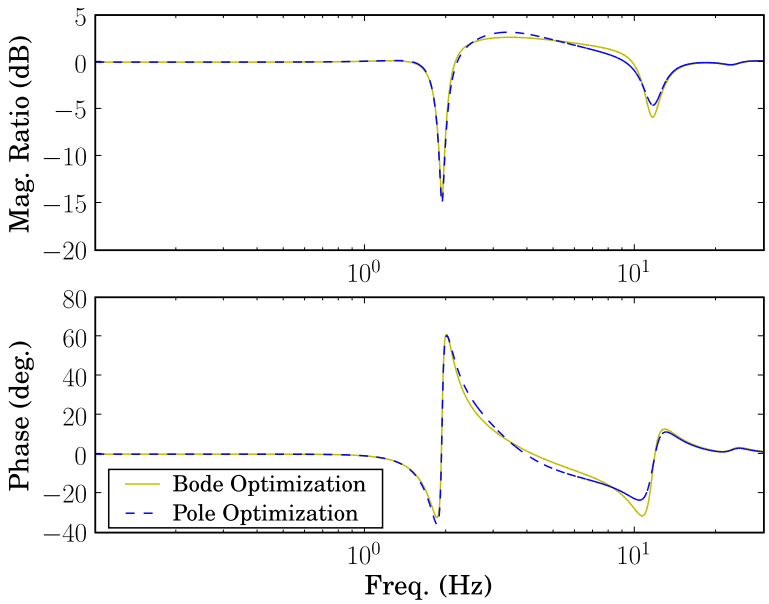

Figure 82: Comparison of disturbance rejection Bode plots $\ddot{x}/d$ for the Bode and pole optimization control designs.

Bode plot $\ddot{x}/d$.

The results of the two optimizations are similar but not identical. Each method uses a cost function written in terms of variables that are natural to its formulation. The Bode optimization tries to maximize the peak amplitude of the first mode of $G_a G_{\text{cl}} G_{\text{flexb}}$ in order to maximize the vibration suppression. The pole-location-based optimization moves the first flexible closed-loop pole as far as possible to the left on the root locus to achieve the same effect (basically maximizing $\omega_1 \zeta_1$). The similarity between the two results confirms that each approach is returning a reasonable design.

The results for the pole optimization are $K_\theta = 0.813$, $K_a = 20.00$, $\omega_c = 14.32$ rad/sec (2.28 Hz). The results for the Bode optimization are $K_\theta = 1.204$, $K_a = 19.99$, $\omega_c = 12.27$ rad/sec (1.95 Hz).

5.4 Generalizing the Control Design Strategy

There are two concerns associated with how to generalize the control design approach presented in this chapter. The first has to do with avoiding a common pitfall in many pole-placement control design strategies. Excellent performance often seems possible in

simulation when the system poles are placed far off to the left on a root locus plot (i.e. all of the system poles are made to have large, negative real parts). These control designs often fail in practice due to actuator saturation and other practical concerns. In order for the control design strategy proposed here to be truly useful, it should include a means of avoiding this pitfall.

The second challenge is generalizing the pole-placement control design strategy used for SAMII, deciding which closed-loop system poles are important and where they should be placed.

5.4.1 Avoiding Saturation

It is important to consider actuator saturation in all practical control design problems. A control system that is well designed for a small step change may perform poorly for a large step change. This is particularly true for systems like SAMII with multiple feedback loops. If SAMII's motion control loop saturates the actuator, the vibration suppression loop will not be able to add its portion of the control signal and the vibration suppression control will be lost. Saturation problems should be investigated by looking at the voltage signal required for a desired closed-loop system response.

The specific control design strategy used for SAMII did not have problems with actuator saturation because the control gains K_θ and K_a were limited. K_a was explicitly limited to 20. K_θ was implicitly limited by the phase margin requirement (in the Bode optimization) or the interaction between the actuator and the structure (in the pole-based optimization).

Figure 83 shows the Bode plot $G_p = \theta/v$. If the motion control compensator is just a proportional controller ($G_\theta = K_\theta$), than K_θ will be limited by the phase margin requirement so that the crossover frequency must be less than approximately 8Hz.

If G_θ is chosen to have the form

$$G_\theta = K_\theta \, G_{\text{cancel}} \tag{202}$$

where

$$G_{\text{cancel}} = \frac{(s^2 + 2\,\zeta_z\,\omega_z\,s + \omega_z^2)\,\omega_p^2}{(s^2 + 2\,\zeta_p\,\omega_p\,s + \omega_p^2)\,\omega_z^2} \tag{203}$$

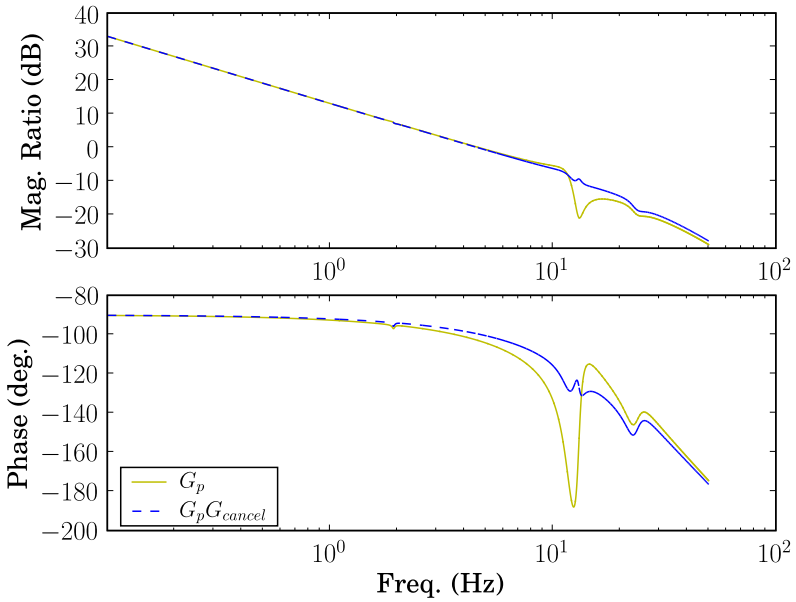

Figure 83: Loop transfer functions for G_θ design.

than ω_z, ζ_z, ω_p, and ζ_z can be chosen to approximately cancel the dynamics that cause the phase dip near 10Hz in G_p. The plot of $G_p\, G_{\text{cancel}}$ in Figure 83 shows the results of this approximate cancellation of the dynamics.

Figure 84 shows the closed-loop actuator Bode plot θ/θ_d for various values of K_θ when $G_\theta = K_\theta\, G_{\text{cancel}}$. It would appear that increasing K_θ increases the band width of the system without any obvious adverse effects.

The adverse effects are apparent in Figure 85, where the corresponding Bode plots for actuator voltage v/θ_d are plotted. A horizontal line is drawn at 20 dB on the magnitude plot. The actuator would saturate while trying to track a sine wave for the desired joint angle θ_d with amplitude 1° at frequencies where the magnitude of v/θ_d is greater than 20 dB. This criterion could be used to limit the actuator gain $K_\theta \leq 2$ for the controller form $G_\theta = K_\theta\, G_{\text{cancel}}$. Other criteria for finding a maximum gain could also be used, but actuator saturation should be explicitly considered in some way.

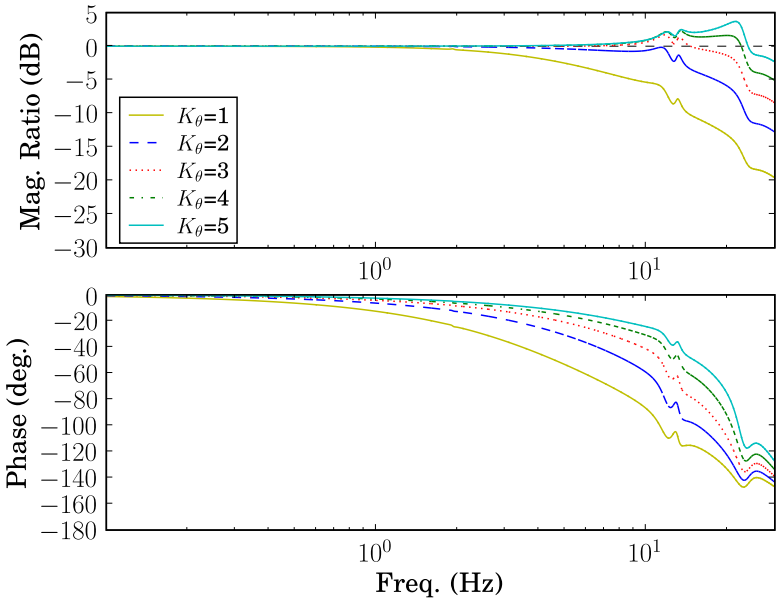

Figure 84: Closed-loop actuator Bode plots θ/θ_d with $G_\theta = K_\theta\, G_{\text{cancel}}$.

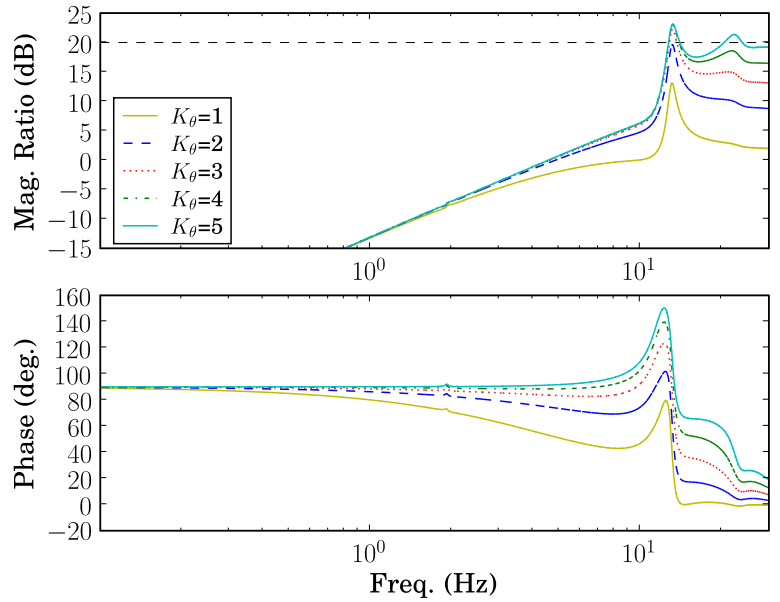

Figure 85: Closed-loop Bode plots for the actuator voltage v/θ_d with $G_\theta = K_\theta\, G_{\text{cancel}}$.

5.4.2 Pole-Placement Strategy

The pole-placement strategy used for SAMII was based primarily on experimental insight into the behavior of the system. A truly general pole-placement strategy for control of flexible structures would be very difficult to formulate. However, some general guidelines can be proposed.

The first problem in formulating a pole-placement strategy is figuring out which poles dominate the dynamics of the system. These will likely be the ones closest to the origin of a root locus, though any unstable pole will be significant. The smallest subset of poles that can reasonably approximate the response of the system to a specified input could also be used to find the dominant poles. A step input could be a good candidate.

Another approach to finding the dominant poles would be to examine the Bode plots over the frequency range of interest to the modeler. This is likely related to the frequency response characteristics of the sensors and actuators. For SAMII, this is the range from 0.1–30 Hz. Once the frequency range of interest has been determined, the modeler can determine which poles are necessary to recreate the Bode plots over this frequency range. The open-loop Bode plot for SAMII's actuator (Figure 57) clearly has a pole at the origin based on the 20 dB/decade roll-off and –90° phase at low frequencies. Acceleration Bode plots like Figure 63 clearly show two peaks in the frequency range 0.1-30 Hz. This means that the two flexible modes in this frequency range are important. Lastly, the filter pole is important because it is introduced by the control scheme.

Once the dominant poles are known, reasonable desired closed-loop locations for them must be specified. Reasonable locations could be found by performing root locus analysis for each gain over the ranges for the gains that were found in the saturation analysis of section 5.4.1. The closed-loop pole locations could also be tracked over a coarse grid in the parameter space. Basically, the sensitivity of each pole to changes in the control gains needs to be assessed so that reasonable goals for moving the poles can be set up.

Problems may occur if the controller gains change significantly or if the control scheme tries to move the poles a significant distance from their starting positions. In those cases, poles that were previously ignored may become dominant. Extra care must be taken to

verify that the closed-loop system behaves as desired.

The pole-placement strategy could be summarized as follows:

- Determine the dominant poles
- Design a controller that moves those poles to desired locations
- Check the closed-loop system to verify the response
- Inspect the closed-loop response for actuator saturation problems

5.5 *Dependence on* fmin

The top level control design algorithm depends on fmin for its optimization. fmin is based on a Nelder-Mead simplex algorithm [46]. The algorithm minimizes a function of n variables by creating a polygon with $n+1$ vertices (this polygon is called a simplex). The minimization takes place as the algorithm moves through the parameter space replacing the "worst" vertex of the polygon. Through this process of replacing one vertex, the polygon grows, shrinks and moves through the parameter space. The algorithm does not use derivatives in its calculations, either explicitly or implicitly.

Even though the Nelder-Mead algorithm has been around for 40 years, proof of its convergence only exists for simple problems of 1 or 2 parameters. There are some known convergence problems, such as following a line in the parameter space that is orthogonal to the gradient [42]. Improving the algorithm is an area of active research [33]. The algorithm is widely used in science and engineering.

The Nelder-Mead algorithm is well suited to this problem because it does not use derivatives and because it makes fairly small changes in the parameters as it moves through the parameter space. Small parameter changes make pole tracking much easier.

While the current control design algorithm depends on fmin, this does not need to remain the case. Through symbolic implementation of the TMM, this work gets the control design problem in a form where numeric search algorithms can be used. Once in that form, these control design problems can provide an interesting application for researchers working on minimization techniques such as genetic algorithms and improvements to Nelder-Mead.

fmin is also used in the system identification portion of this thesis, where the optimization is searching for 8–10 system parameters simultaneously. The algorithm converged reliably in that work as well.

5.6 Robustness

Figures 86–103 investigate the robustness of the control design to errors in the estimates of the unknown system parameters k_{base}, c_{base}, k_{joint1}, c_{joint1}, K_{act}, p_{act}, ks_{act}, c_{act}, and $\text{Gain}_{\text{bode1}}$. These are the parameters determined by system identification. This analysis could be extended to any and all parameters in the model. The Bode plots show how the closed-loop system performance will be affected if the actual system parameters are different from the values used in the model for control design. The analysis was performed by substituting into the system model values for the various parameters that differ from the nominal values by $\pm 10\%$.

Figures 86 and 87 show that k_{base} mainly affects the first mode of the system. k_{base} is the stiffness coefficient for the torsional spring/damper at the top of Figure 104, representing the fact that the beam is not perfectly clamped. (The system identification procedure and results will be discussed in detail in chapter 7.) As expected, if k_{base} is lower than the nominal value, the first mode shifts to a slightly lower frequency. If k_{base} is higher than the nominal value, the opposite is true.

Figure 88 and 89 show that a 10% change in c_{base} has very little affect on the closed-loop response of the system. c_{base} is the coefficient of the damper that is in parallel with k_{base}.

Figures 90 and 91 show that k_{joint1} affects the second mode in a very similar way to how k_{base} affects the first mode. k_{joint1} represents compliance in joint 1.

Figure 92 and 93 show that a 10% change in c_{joint1} has very little affect on the closed-loop response of the system. c_{joint1} is the coefficient of the damper that is in parallel with k_{joint1}.

Figure 94 and 95 show the affects of changes in the actuator gain K_{act} on system performance. The changes make intuitive sense: if the gain is higher than the nominal value, the response amplitude increases. The opposite is true if the value is smaller than

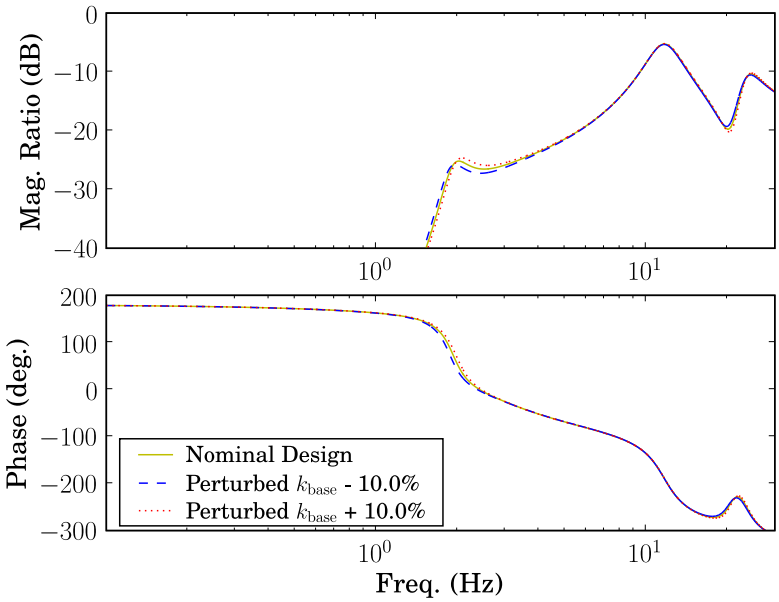

Figure 86: Bode plot for $\ddot{x}/\hat{\theta}_d$ investigating the robustness of the control design to changes in k_{base}.

nominal. These effects are less pronounced at lower frequencies where the closed-loop control is able to compensate for inaccuracies in the model.

Figures 96 and 97 show the affects of changes in the actuator pole location p_{act}. Again the changes seem correct. If p_{act} is lower than expected, the Bode plots start rolling off at a slightly lower frequency. If p_{act} is higher than nominal, the roll off starts at a higher frequency.

Figures 98–101 show that 10% changes in the spring and damper coeffiecents for the actuator compliance (ks_{act} and c_{act}) do not significantly impact the system response.

Figure 102 and 103 show the affects of changes in the gain of the $\ddot{x}$ Bode output.

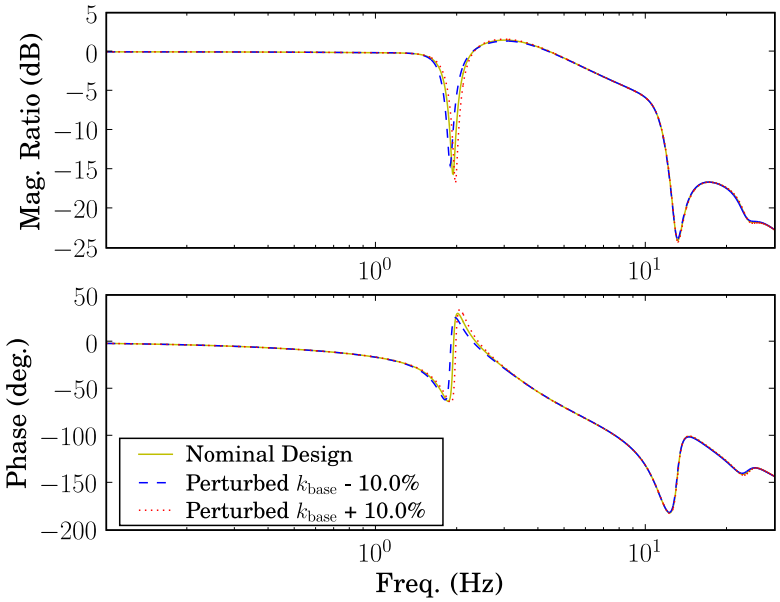

Figure 87: Bode plot for $\theta/\hat{\theta}_d$ investigating the robustness of the control design to changes in k_{base}.

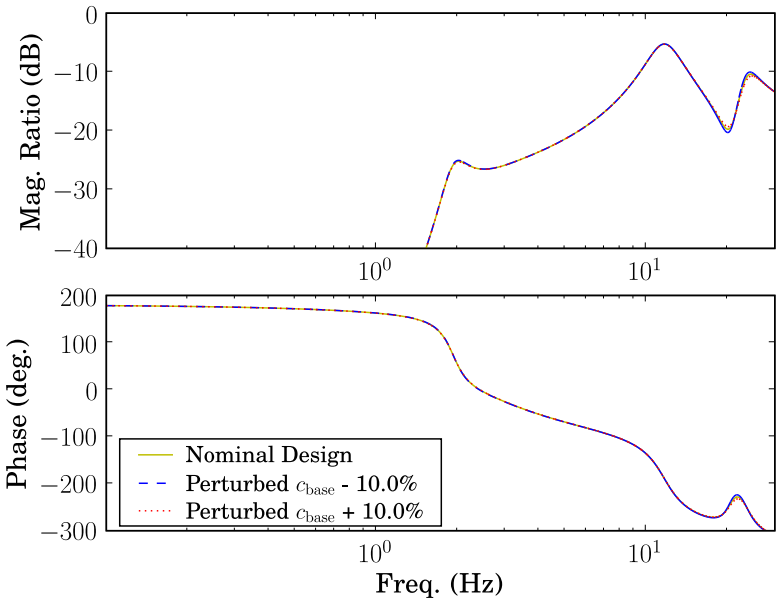

Figure 88: Bode plot for $\ddot{x}/\hat{\theta}_d$ investigating the robustness of the control design to changes in c_{base}.

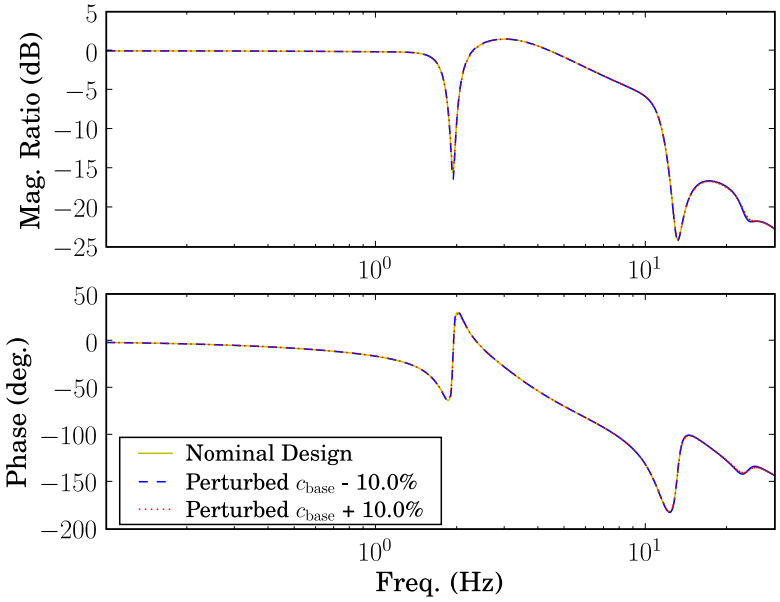

Figure 89: Bode plot for $\theta/\hat{\theta}_d$ investigating the robustness of the control design to changes in c_{base}.

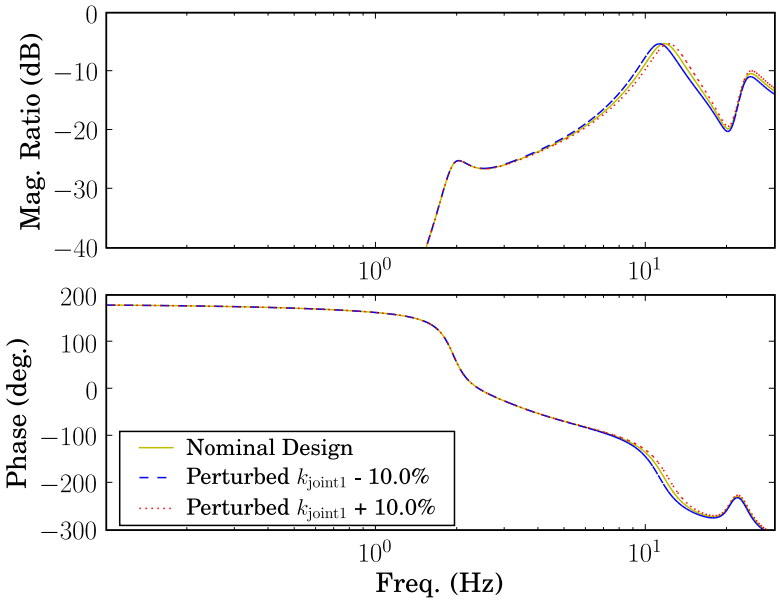

Figure 90: Bode plot for $\ddot{x}/\hat{\theta}_d$ investigating the robustness of the control design to changes in k_{joint1}.

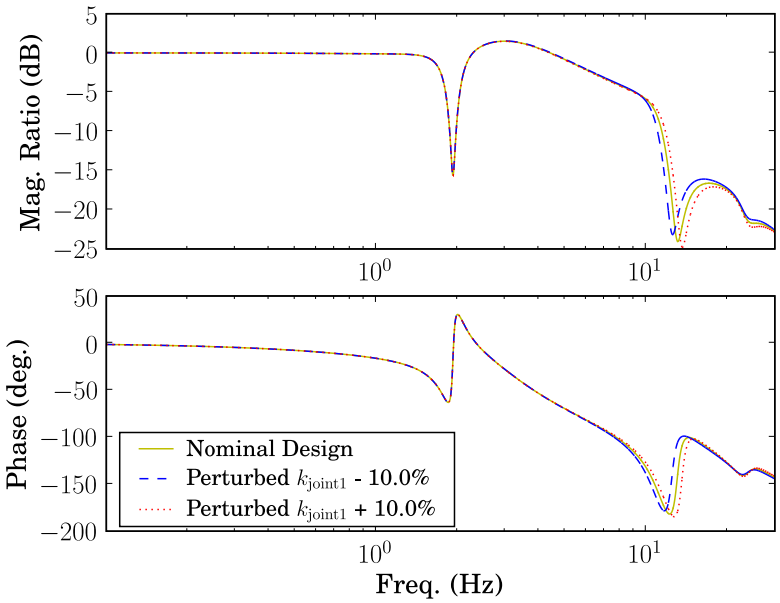

Figure 91: Bode plot for $\theta/\hat{\theta}_d$ investigating the robustness of the control design to changes in k_{joint1}.

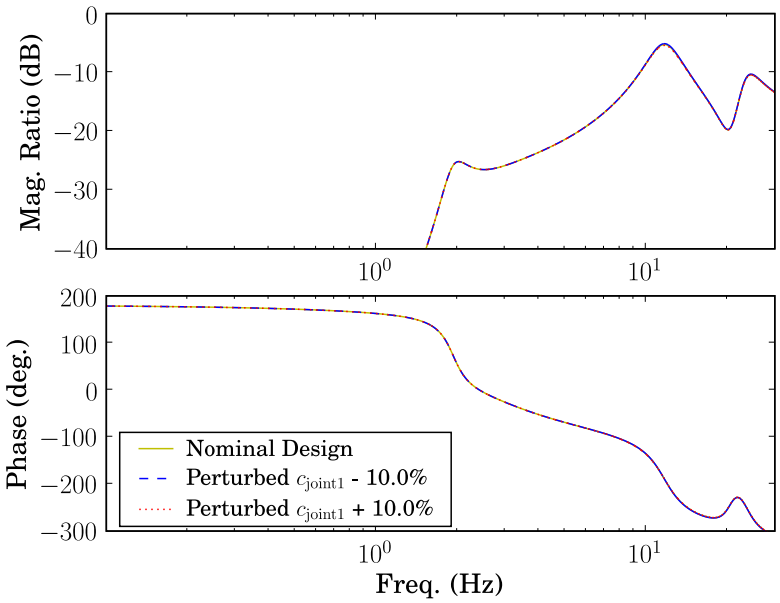

Figure 92: Bode plot for $\ddot{x}/\hat{\theta}_d$ investigating the robustness of the control design to changes in c_{joint1}.

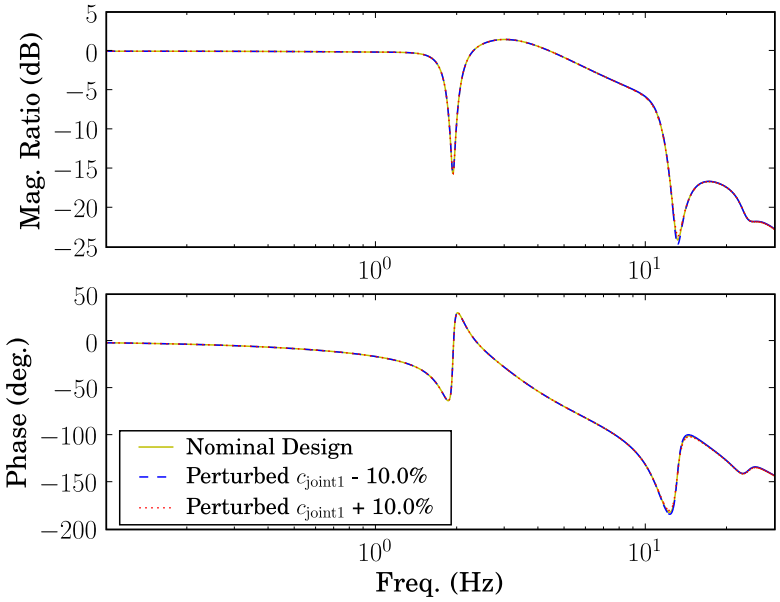

Figure 93: Bode plot for $\theta/\hat{\theta}_d$ investigating the robustness of the control design to changes in c_{joint1}.

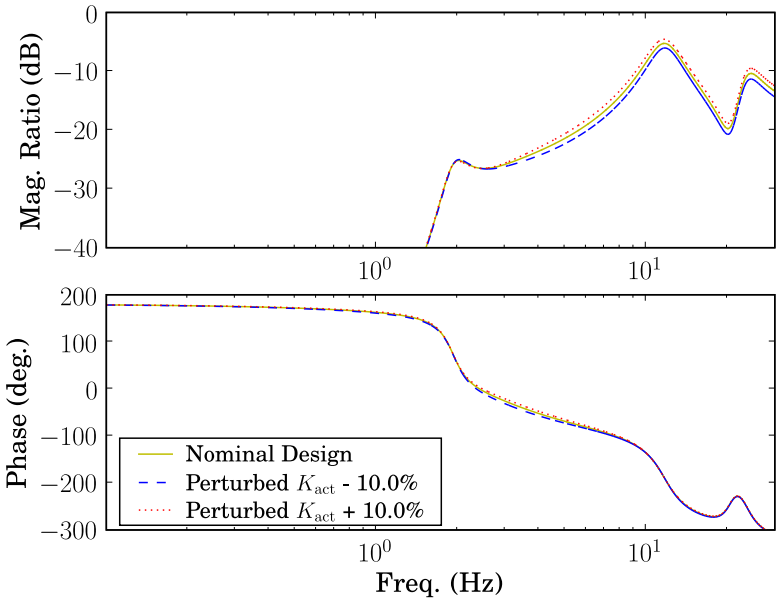

Figure 94: Bode plot for $\ddot{x}/\hat{\theta}_d$ investigating the robustness of the control design to changes in K_{act}.

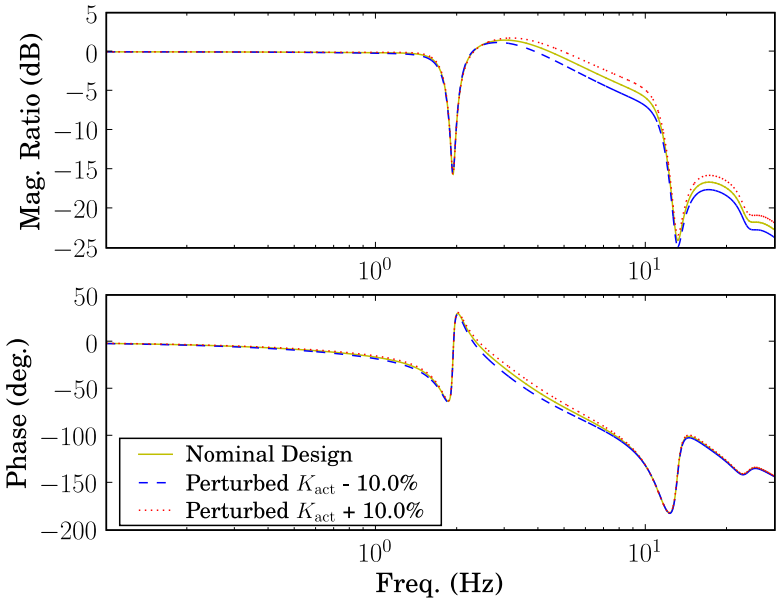

Figure 95: Bode plot for $\theta/\hat{\theta}_d$ investigating the robustness of the control design to changes in K_{act}.

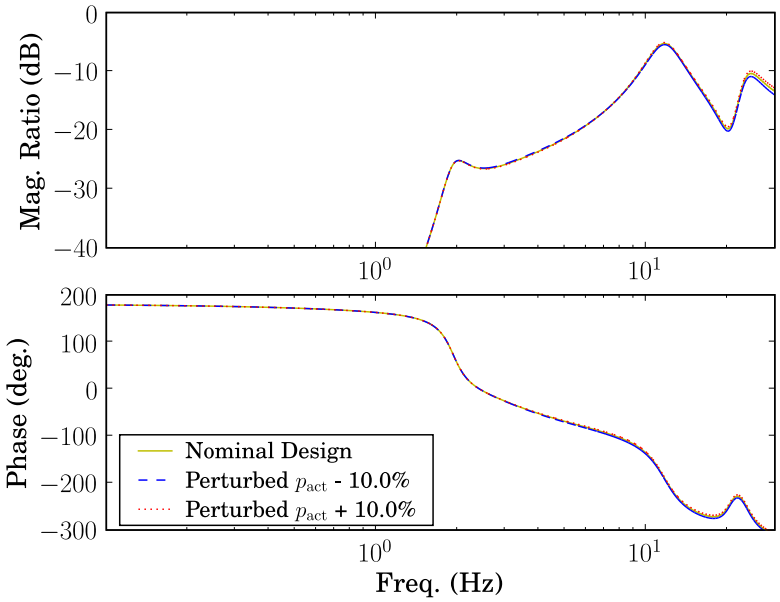

Figure 96: Bode plot for $\ddot{x}/\hat{\theta}_d$ investigating the robustness of the control design to changes in p_{act}.

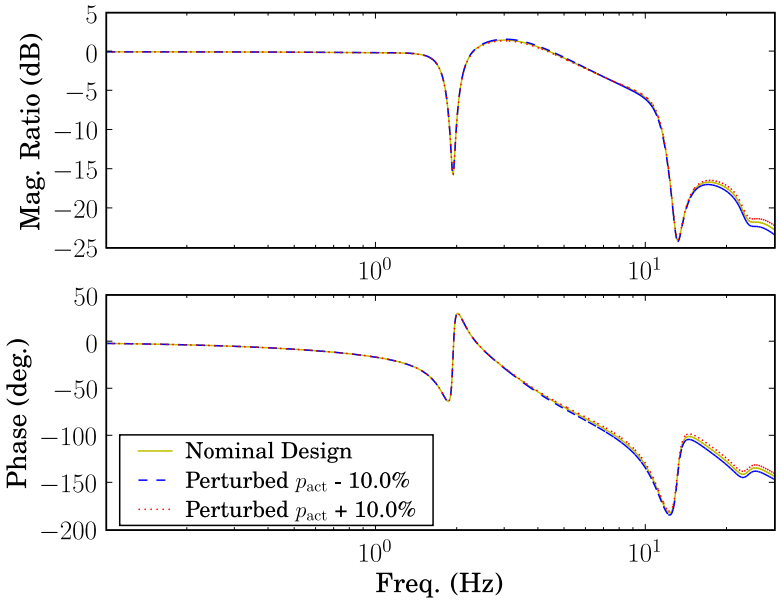

Figure 97: Bode plot for $\theta/\hat{\theta}_d$ investigating the robustness of the control design to changes in p_{act}.

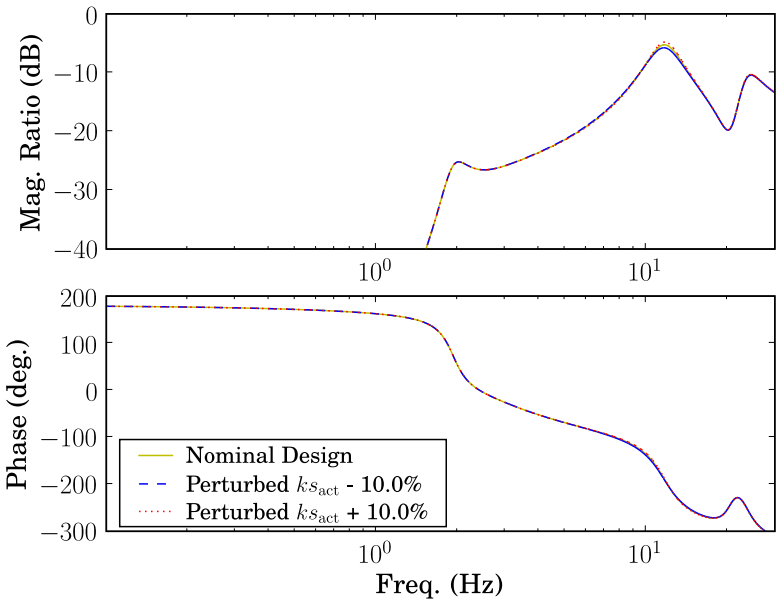

Figure 98: Bode plot for $\ddot{x}/\hat{\theta}_d$ investigating the robustness of the control design to changes in ks_{act}.

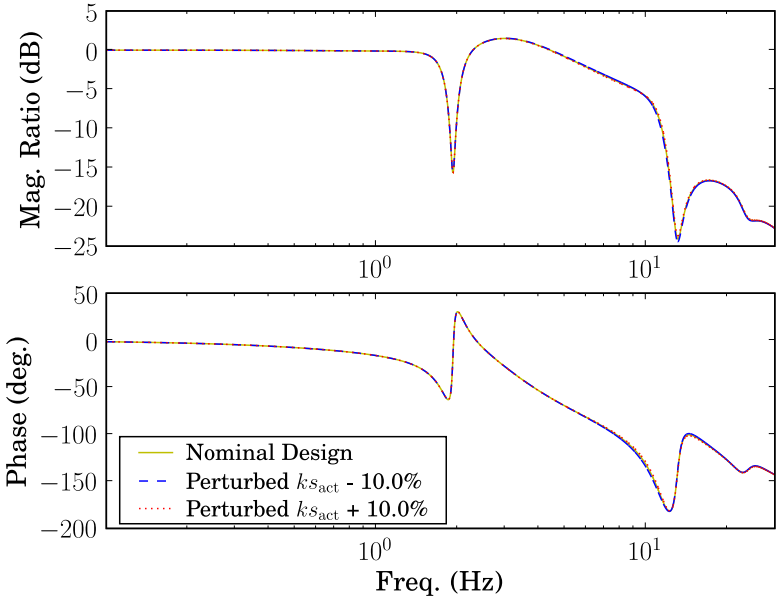

Figure 99: Bode plot for $\theta/\hat{\theta}_d$ investigating the robustness of the control design to changes in ks_{act}.

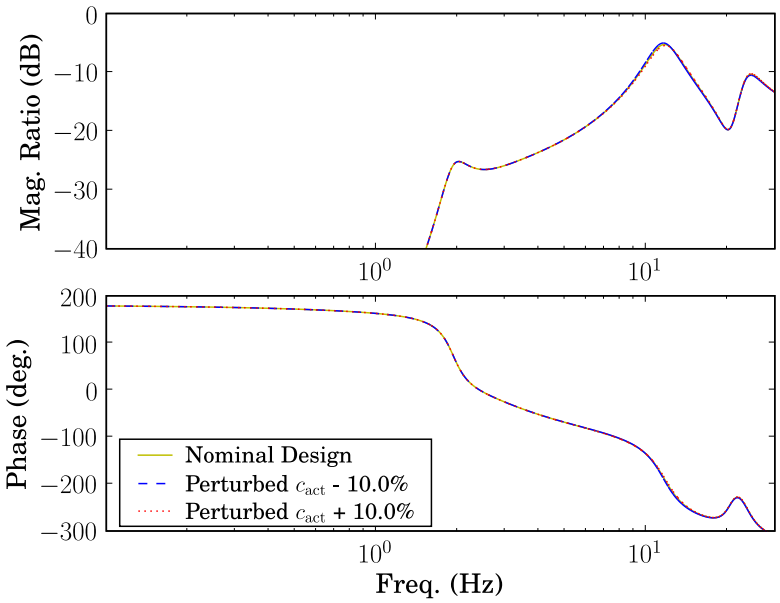

Figure 100: Bode plot for $\ddot{x}/\hat{\theta}_d$ investigating the robustness of the control design to changes in c_{act}.

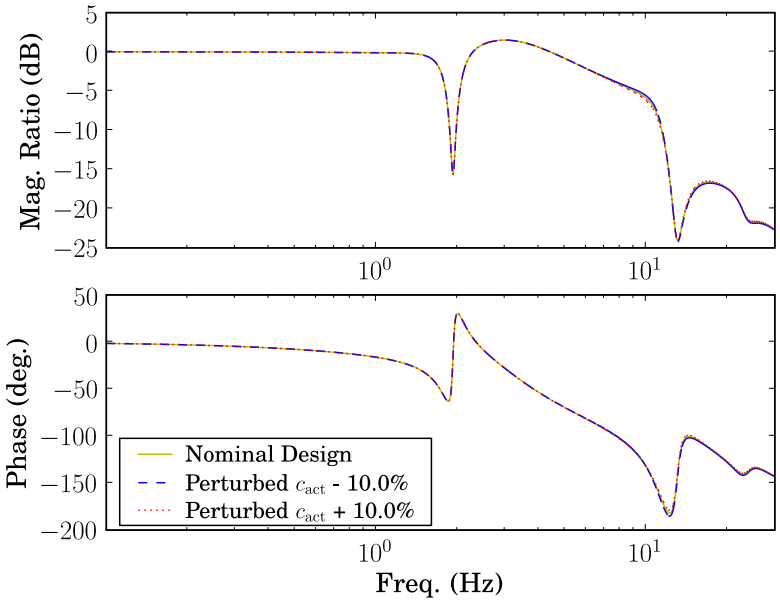

Figure 101: Bode plot for $\theta/\hat{\theta}_d$ investigating the robustness of the control design to changes in c_{act}.

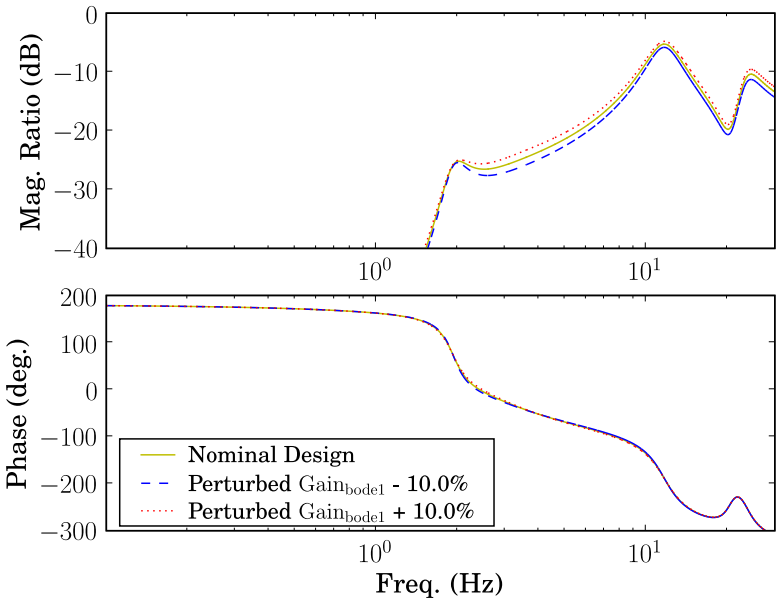

Figure 102: Bode plot for $\ddot{x}/\hat{\theta}_d$ investigating the robustness of the control design to changes in $\text{Gain}_{\text{bode1}}$.

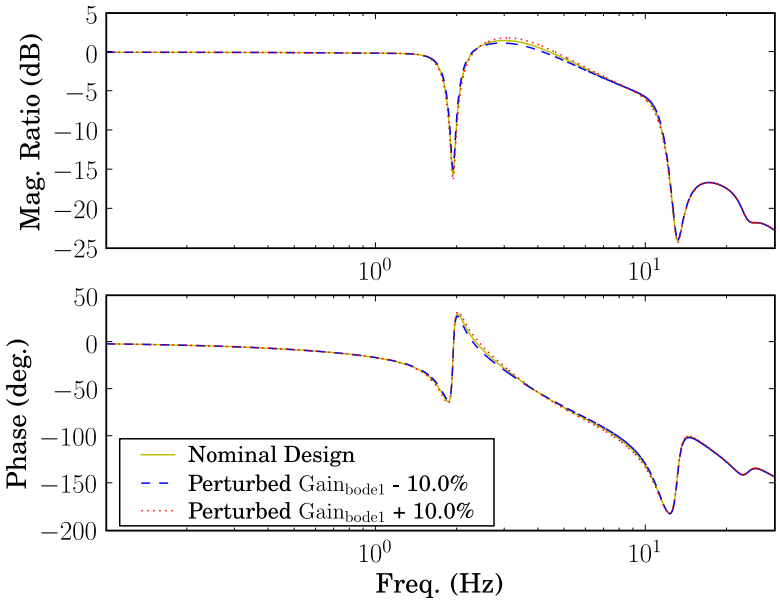

Figure 103: Bode plot for $\theta/\hat{\theta}_d$ investigating the robustness of the control design to changes in $\text{Gain}_{\text{bode1}}$.

CHAPTER VI

SOFTWARE DESIGN

6.1 Introduction

One contribution of this work is the design and creation of a TMM analysis software package. This package is controls focused and object oriented. It is easy to use and makes powerful analysis and design capabilities available to new users who are not experts in the TMM. This package is also designed to be user extensible. Users can add to the existing modeling and analysis capabilities by creating new transfer matrix element types or by adding functions to the TMM system class.

The code is designed in a modular fashion so that it is easy to navigate. This not only makes the details easily accessible to TMM experts, it is also practically necessary for user extensibility. Users are not going to invest their time adding to disorganized code they cannot understand.

Specifically, the TMM analysis package is written as a module in the Python programming language, making use of several excellent suites of Python tools including NumPy, SciPy, Matplotlib, and Mayavi.

6.1.1 Overview

The main idea driving software design is making all of the functionality and power of the TMM available in an easy to use package. This should be done in a way that allows novices and users who are not experts in the TMM to use it effectively while still making the details accessible to TMM experts. The software allows vibrations analysis to be done by determining natural frequencies and mode shapes of the system. The response to forcing can also be analyzed through Bode plots and operating deflection shapes. Controls analysis and design can be done by generating open or closed-loop Bode plots, root loci, and symbolic transfer functions as well as determining control gains through the optimization routines

discussed in chapter 5.

Functionally, the software is mainly composed of two classes that are used to model flexible robots or structures using the TMM: TMMElement and TMMSystem. The TMMElement is used to describe the pieces of a structure or robot such as beam elements, rigid bodies, actuators, springs, and dampers. The TMMSystem is made up of a list of TMMElement's along with system boundary conditions and a list of Bode outputs.

6.1.2 Software Usage

Before discussing the design of the individual pieces of the software, it will be beneficial to have a clear understanding of how the pieces fit together.

The first step in using the software is creating TMM element models for each component of the system. For a flexible robot, these components will include things like beam elements for flexible links, torsional spring elements for joint compliance, rigid body elements for rigid links, and actuator elements of some kind. If a software model of for each element type does not already exist, the user will need to create one by deriving from the TMMElement base class as discussed in section 6.5.

Once software models for each element have been created, those elements are assembled into a transfer matrix system model or TMMSystem. A TMMSystem is created from a list of TMMElement's, system boundary conditions, and a list of the Bode outputs that are of interest. These Bode outputs specify measured output signals that the model should simulate.

Once the TMMSystem model has been created, all of the powerful analysis techniques built-in to the software are accessible. These include capabilities for Bode analysis, finding system natural frequencies and mode shapes, system identification, symbolic analysis, and control design and optimization. All of these analysis and design tools require only a few lines of code beyond the definition of the TMMSystem model.

6.1.3 Novel Features

There are several aspects of the software design that are novel. The primary one is that it is object-oriented. This brings the TMM into the realm of modern programming and brings

about several practical benefits. User extensibility, inheritance, encapsulation, and clean and clear syntax are all benefits that depend on the object oriented nature of the software. Part of the contribution of this thesis in the area of software design is the creation of the TMMElement and TMMSystem classes that form the foundation of the software package.

The object-oriented implementation is an important part of the software design primarily because it facilitates user-extensibility. The object-oriented framework and specifically inheritance make it straightforward for a user to create a new transfer matrix element.

Other novel features of the software include the ability to easily perform symbolic TMM analysis, control design and optimization tools, and automation of the controls engineer's work flow through integrated system identification capabilities.

Providing a way to do TMM analysis symbolically enables control design. Closed-form expressions for the closed-loop transfer functions can be obtained for any system the TMM can model, including those with continuous elements. If the system includes continuous elements, these transfer functions will be infinite dimensional. These symbolic transfer functions facilitate pole-placement or optimal control by providing expression which can be used in numeric search algorithms to find the control parameter values (PID gains for example) that place the closed-loop poles at desired locations.

Work flow automation makes the software more than just a modeling and design tool. It starts to become a higher-level language for the controls or vibrations engineer, freeing them from dealing with low-level details and repetitive tasks. This is done by integrating into the software common tasks related to modeling or control design. An example of this work flow automation is building system identification capabilities into the software. Once a general form for a model is determined it is often necessary to do system ID in one form or another to quantify certain unknown system parameters. This can be a labor intensive process with a lot of data to analyze and the possibility for many low-level programming bugs in the data processing scripts. Integrating these capabilities into the software can make the life of the controls or vibrations engineer much easier. However, care must be taken to intelligently automate this process and build in ways for the engineer to check that the automated system is doing the analysis correctly. Integrated system ID will be discussed in

chapter 7.

6.2 The Benefits of an Object-Oriented Approach

Using an object-oriented approach adds significant value to this work. The object-oriented framework leads to code that is clear, clean, user extensible, and easy to reuse. The code is clear because conceptually a transfer matrix element is an object that has certain properties and needs to do certain things.

Depending on the background of the reader, the term object-oriented programing may be somewhat unfamiliar. It may seem vague and esoteric, but the basic idea is to describe what you want to do in a sentence and then replace the nouns with classes (objects) and the verbs with methods [39, p. 349].

Following this approach, TMM analysis could be described in a few sentences. Transfer matrix analysis consists of modeling a system by a connection of elements. The analysis finds the Bode response and the natural frequencies and mode shapes of the system. Natural frequencies are found by searching numerically for values of the frequency variable s that cause a characteristic determinant of the system to go to zero. Having described the analysis in words allows the pieces for object-oriented design to be recognized. A TMMSystem consists of a list of transfer matrix elements, a set of system boundary conditions, and possibly some Bode outputs (these are the properties of the class). The TMMSystem will need methods for finding the natural frequencies and mode shapes as well as the Bode response. A TMMElement will have a set of parameters (such as length, modulus, mass per unit length, and second moment of inertia for a beam) and must have a method for calculating its transfer matrix for a given value of s.

Oriented-oriented programing leads to clean code because an object provides a nice container for arbitrarily complicated data structures. This is called encapsulation. As an example, a model of a TMM system that is not object-oriented would include a list of element types and a separate list of parameters for each element. When the system transfer matrix needs to be calculated, each element type would be used to determine which function to call to calculate the correct element transfer matrix and then the parameters for that

element would be passed to that function. A snippet of code might look like this:

```
1       if elementtype == 'Rigid':
2           U = RigidTransferMatrix(s, parameters)
3       elif elementtype == 'Beam':
4           U = BeamTranserMatrix(s, parameters)
```

and so on for all possible element types. RigidTransferMatrix and BeamTranserMatrix are the names of functions that return element transfer matrices for rigid and beam elements. The TMM system code must be modified to include a new **elif** statement any time a new element type is added to the model.

In contrast, the parameters for each element in an object-oriented approach are encapsulated by the element and each element type has its own method for calculating its transfer matrix. This means that the TMMSystem model no longer needs to know the element types explicitly and does not need to pass the parameters to the method. The code for getting the transfer matrix of an element is simply

```
U = element.GetTransferMatrix(s)
```

regardless of element type, and the TMMSystem code does not need to be modified to accommodate new element types (any TMMElement that has a GetTransferMatrix method is valid). This one line of code replaces the entire **if** ... **elif** structure of the non-object-oriented approach which would contain at least 2 lines per possible element type (leading to at least 10-20 lines of code total).

User extensibility is one of the most important aspects of the design of this analysis package. For this analysis package to be truly useful, others must be able to apply it to a broad range of problems. It would be nearly impossible to try to anticipate all possible problems that future users would want to analyze using the TMM and it would be terribly time consuming to code up all of the possible TMMElement's that can be thought up. A straightforward way for future users to create their own elements is essential. Object-oriented programming has also been described as a way to specify set membership [39, p. 346]. Inheritance is one of the primary facets of object-oriented programing and it makes it possible to define increasingly customized or specialized sets. Inheritance refers

to the fact that a derived class automatically inherits all of the methods and properties of the base class. The primary case here is the idea of a TMMElement class. A base class can be defined that has all of the properties and methods of a transfer matrix element and then the users derive more specialized cases from the base class. Examples of classes derived from the TMMElement base class in this work include BeamElement, RigidMassElement, and AngularVelocitySource. TMMElement is a large set including all possible transfer matrix elements and BeamElement is a more specialized subset of the TMMElement set.

As an example of the power and usefulness of inheritance, consider Bode analysis that requires augmented transfer matrices. Augmented transfer matrices can be used to model external inputs to a TMMSystem [49, section 3-5, p. 82]. If the transfer matrix for an element is given by $\mathbf{U}$, then the augmented transfer matrix for an element with no external inputs (an unactuated element) is

$$\mathbf{U}_{\text{aug}} = \begin{bmatrix} & & 0 \\ & \mathbf{U} & \vdots \\ & & 0 \\ 0 & \cdots 0 & 1 \end{bmatrix} \tag{204}$$

GetAugMat could be defined in the TMMElement class and inherited by all new TMMElement's derived from the base class. Only elements that will have external inputs need to override the method from the base class. All unactuated elements will correctly return an augmented transfer matrix with no effort on the part of the user, because the method is inherited from the base class. This means that Bode analysis (which typically depends on augmented transfer matrices to provide the input) can proceed cleanly and easily once GetAugMat has been written for the actuator element (AngularVelocitySource in the case of SAMII). In contrast, a non-object-oriented approach would likely require that each element have a function that calculates its augmented transfer matrix or an augmented option be added to all existing TMM functions. This is an example of how object-oriented code is easier to maintain than non-object-oriented code. Methods that are inherited from the base class only need to be changed or added in one place.

Object-oriented programing leads to reusable code because an existing solution can

easily be adapted to a new problem by inheriting a new class from an existing one that already has some (or many) of the desired features.

6.3 The Choice of Python

It is difficult to anticipate what the consequences of this choice will be in a few years, but at the time of this writing Python seemed to be by far the best choice for this work. Initial work was done in Matlab©, and in theory all of the analysis could be done in Matlab© or any other language capable of working efficiently with matrices. But at the time this work switched from Matlab© to Python, Matlab© was not capable of truly doing object-oriented programing. Python has several other strengths that make it attractive:

- It is a general purpose programming language, well suited for any programming task – not just mathematical analysis (it is packages like SciPy and NumPy that give Python its linear algebra capabilities).

- It is an interpreted and dynamically typed language. The user writes scripts that do not need to be compiled and variables do not need to have their types or sizes declared ahead of time.

- It is module focused. Modules provide a clean way to compartmentalize code that is related to a certain task or idea (like TMM analysis). A module may contain any number of functions and classes as well as sub-modules. Python module design prevents one module from overwriting the names, functions, or classes of another (if I define a `sqrt` function in a module called `myfunctions` and import that module, the built-in function called `sqrt` is not overwritten – and my new function is accessible by calling `myfunctions.sqrt`).

- Modules for linear algebra, plotting, optimization, and many other topics already exist (there are literally hundreds of Python modules available for all kinds of programing tasks, including many of engineering interest).

- It is an easy language to program in. This is a hard claim to quantify or convince

others of, but coding in Python seems to go faster with fewer bugs than in any other language (in the opinion of the author).

- It is cross platform.
- It is free and open source.

As one example of how Python is an easy an elegant language to code in, consider a case where the elements of a vector need to be iterated over. In many languages this task requires explicitly declaring an index:

```
for i in length(vector),
    element=vector[i]
    (do something with element)
```

In Python, this iteration is cleaner:

```
for element in vector:
    (do something with element)
```

This does much more than eliminate the need for one extra line of code, it gets rid of a low-level coding detail and the possibility for bugs that go along with it.

6.4 Example

Several of the claims of this thesis related to software design can be seen by looking at the full TMM model for SAMII. A picture of SAMII and a schematic representation are shown in Figure 104. The corresponding code for this model is

```
1 def olsamiimodel():
2     basespring=TorsionalSpringDamper4x4({'k':166358.0,'c':468.789},
          symlabel='base', unknownparams=['k','c'])
3     beam=samiiBeam()
4     link0=samiiLink0()
5     j1spring=TorsionalSpringDamper4x4({'k':4028.28,'c': 6.3058},
          symlabel='j1', unknownparams=['k','c'])
6     link1=samiiLink1()
7     avs=AngularVelocitySource4x4({'K':0.435489,'tau':173.833}, symlabel=
          'act', unknownparams=['K','tau'])
8     j2spring=TorsionalSpringDamper4x4({'k':1900.49,'c':21.6805},
          symlabel='j2', unknownparams='all')
```

```
9       bodeout1=bodeout(input='j2v', output='a1', type='abs', ind=beam,
            post='accel', dof=0, gain=0.35, gainknown=False)
10      bodeout2=bodeout(input='j2v', output='j2a', type='diff', ind=[
            j2spring, link1], post='', dof=1, gain=180.0/pi)
11      link2=samiiLink2()
12      return ClampedFreeTMMSystem([basespring, beam, link0, j1spring,
            link1, avs, j2spring, link2], bodeouts=[bodeout1,bodeout2])
```

The entire model takes only 12 lines of code (not including the header portion). This code is clean and simple, yet fully functional, because of the object-oriented nature of the design. It also serves to illustrate inheritance and encapsulation.

basespring, beam, link0, j1spring, link1, avs, and j2spring are all objects that inherit from TMMElement. They are all software representations of the corresponding physical elements of the model in Figure 104. Because they inherit from TMMElement, they will all have many of the same properties and methods.

Lines 2, 5, and 8 create TorsionalSpringDamper objects to model compliance in the joints and the connection to the beam at the top of Figure 104. Line 3 creates the beam element. Lines 4, 6, and 11 create RigidMassElement's representing links 0, 1, and 2. Line 7 creates the AngularVelocitySource element that models the actuator. Lines 9 and 10 define the Bode outputs, which are a means of specifying sensor locations and types so that model output can be compared directly with measured Bode responses. Line 12 creates the TMMSystem object by passing in an ordered list of TMMElements and a list of bodeouts. ClampedFreeTMMSystem is derived from TMMSystem. As implied by the name, ClampedFreeTMMSystem specializes TMMSystem by specifying the boundary conditions.

The TMMElement objects are also examples of encapsulation. By looking at the output of basespring.__dict__, all of the information stored in that object is visible:

```
1 {'elemstr': 'spring',
2  'elemtype': 4,
3  'functionparams': None,
4  'label': '',
5  'maxsize': 4,
6  'params': {'k': array([ 166358.]), 'c': array([ 468.789])},
7  'symlabel': 'base',
8  'symname': 'Ubase',
9  'symparams': {'k': ['kbase'], 'c': ['cbase']},
```

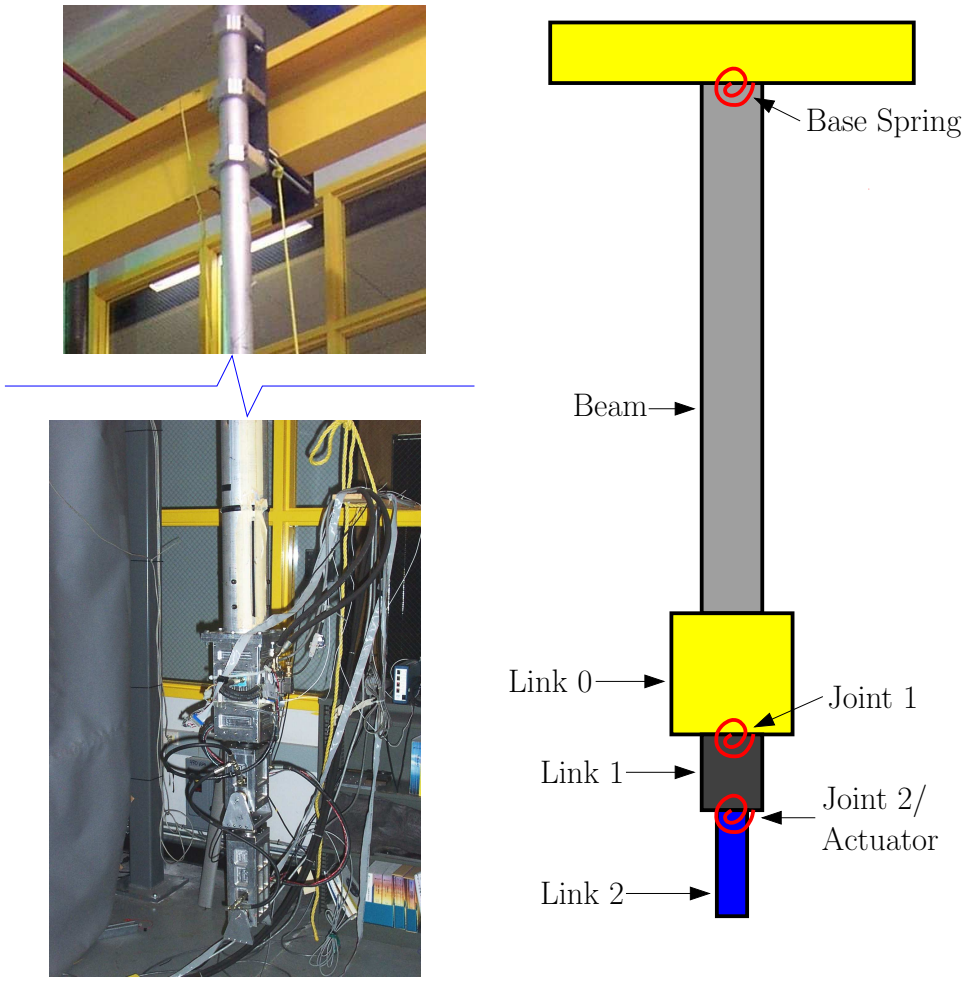

Figure 104: Picture and schematic of SAMII used to illustrate the TMM model.

```
10   'symsub': False,
11   'unknownparams': ['k', 'c']}
```

All of this information is passed to the TMMSystem class simply by including basespring in the list of elements in the last line. The properties of basespring include the numeric values for the spring and damping ratio, names and labels used in symbolic analysis, a list of parameters to be treated as unknown during system identification, and other details.

The labeling of some of the element parameters as unknown in the code for olsamiimodel alludes to the fact that system identification capabilities are integrated into the software. This will be discussed in chapter 7.

The SAMII model depends on two other classes besides the TMMElement class: bodeout and TMMSystem (ClampedFreeTMMSystem is derived from TMMSystem). These three classes will be discussed in the following sections.

6.5 Design of the TMMElement Class

6.5.1 Purpose/Overview

The TMMElement class is the primary building block of the modeling portion of the software. In the same way that defining a transfer matrix for an element is the first and most important step in using the TMM, deriving a new TMMElement is the first and most important step in using this software package. The TMMElement is an abstraction and generalization of all of the things that TMM elements must do and the properties they have.

The TMMElement class is also the primary means for user extensibility. A user can give the software the ability to model new kinds of elements by deriving from the TMMElement class. One simple example would be defining the transfer matrix for a new actuator. A more complicated example would be deriving several new TMMElement's for thermal or electrical systems and using the software to analyze new kinds of modeling and control problems. Provided that the user was still concerned primarily with Bode plots, system eigenvalues, and system eigenvectors, all the existing analysis tools can still be used and the user only needs to define the new TMMElement's.

The design of the TMMElement class consists mainly of answering two questions (as

alluded to in section 6.2): what must a TMMElement do? (these are the methods) and what properties does it have? The TMMElement class can also be thought of as the most general element that could be used in TMM modeling. In object oriented programming, inheritance is a means of specialization. All classes that derive from TMMElement will be more specialized than the base class. The base class is the most general class and describes properties and methods that all the derived classes will share. It can also be thought of as providing default methods and properties that will work for most elements, but that will be overridden by some. An example of this is the definition of a method to get the augmented transfer matrix of an unactuated element as described in equation (204). Defining this method in the base class means that new elements will have this method by default. For all unactuated elements this is perfect and saves the author of any new element classes from having to write such a method. However, the method will have to be overridden in the case of actuated elements.

6.5.2 Methods

The main thing a TMMElement must do is return its transfer matrix given a value of s as an input. There will need to be methods for doing this numerically and symbolically as well as in regular and augmented form (where the augmented form is used to find the forced response of the system). Another thing a TMMElement must do is return a homogeneous transformation matrix for 3D visualization purposes.

6.5.3 Properties

The properties of a TMMElement will vary based on what type of element it is, but they will basically be whatever parameters are needed to calculate the transfer matrix and the homogeneous transformation matrix. For example, a beam element will have properties corresponding to length L, stiffness EI, and mass per unit length μ. All TMMElement's will also have properties related to details about symbolic analysis and which parameters are unknown when it comes to system identification.

6.5.4 Code/Usage

The majority of the code for implementing the TMMElement base class is shown below. There are a few other methods that implement lower-level details that have been omitted because they are not relevant to this discussion. The full code is described in more detail in appendix C and can be downloaded from http://www.imdl.gatech.edu/ryan/thesis/thesis.htm.

```
1  class TMMElement:
2      def __init__(self,elemtype,params,maxsize=12,label='',symname='',
           symlabel='',symsub=False,unknownparams=None,functionparams=None)
           :
3          self.elemtype=self.parsetype(elemtype)
4          self.params=params
5          self.maxsize=maxsize
6          self.label=label
7          self.symlabel=symlabel
8          self.symname=symname
9          self.symsub=symsub
10         self.unknownparams=unknownparams
11         if functionparams is not None:
12             if not shape(functionparams):
13                 functionparams=[functionparams]
14         self.functionparams=functionparams
15         if isinstance(elemtype,str):
16             self.elemstr=elemtype
17         else:
18             self.elemstr=''
19
20     def GetMat(self, s):
21         raise NotImplementedError
22
23     def GetAugMat(self,s):
24         N=self.maxsize
25         tempout=eye(N+1,dtype='D')
26         tempout[0:N,0:N]=self.GetMat(s)
27         return tempout
28
29     def GetSymMat(self):
30         s=symstr('s')
31         symmat=self.GetMat(s,sym=True)
32         return SymstrMattoMaxima(symmat,self.symname)
33
```

```
34      def GetAugSymMat(self):
35          s=symstr('s')
36          symmat=self.GetAugMat(s,sym=True)
37          return SymstrMattoMaxima(symmat,self.symname)
38
39      def GetMaximaLines(self,aug=False):
40          """This is the method for symbolic analysis based on GetSymMat
               and GetAugSymMat.  This function outputs a list that can be
               appended to a latexlist as part of the input to the Python-
               Maxima-Latex symbolic engine."""
41          mylist=[]
42          out=mylist.append
43          ws='\t'
44          out('\\begin{maxima}')
45          if self.symname[0]=='U':
46              mylabel='\\U{'+self.symname[1:]+'}'
47          else:
48              mylabel='\\M{'+self.symname+'}'
49          out(ws+"\\parseopts{lhs='"+mylabel+"'}")
50          if aug:
51              out(self.GetAugSymMat())
52          else:
53              out(self.GetSymMat())
54          out('\\end{maxima}')
55          return mylist
56
57      def GetHT(self):
58          raise NotImplementedError
```

Briefly for those not familiar with Python syntax, line 1 declares a new class named TMMElement. Lines 2–18 define the __init__ method which describes how a new instance of the TMMElement gets created or initialized. All methods of a class have self as the first argument which refers to the class instance for which the method is called. Lines 20–21 define the GetMat method which takes s as an input and should return the transfer matrix for the element evaluated at s. The base TMMElement class is not intended to be used directly, new elements must be derived from it and they must override the GetMat method. As a result, the GetMat method in the base class does nothing but raise an error.

Lines 23–27 define the GetAugMat method which adds the additional row and column to the element transfer matrices for unactuated elements. It does this by creating an identity

matrix of dimension N+1 (line 25) and then substituting the results from GetMat into the upper left N×N portion of the matrix (line 26).

Lines 29–55 form the basis of symbolic TMM analysis. GetSymMat and GetMaximaLines use the symbolic output of GetMat to generate an input script for Maxima. Maxima is the computer algebra program used with the Python TMM analysis package. GetMaximaLines (lines 39–55) is the primary function for symbolic analysis of a TMMElement. It creates a list of text lines that can be saved to an input file for Maxima.

The GetSymMat serves only as a means for GetMaximaLines to call GetMat while requesting symbolic output. GetMaximaLines could call GetMat directly itself, but the GetSymMat method provides a means for the user to override how the symbolic TMM matrix is created if there is some reason why the numeric and symbolic transfer matrix cannot be made to come from the same function.

Lines 57–58 are a place holder for GetHT, which stands for get the homogeneous transformation matrix associated with the element. GetHT is another method that must be overridden. It is used in the three dimensional visualization portion of the software. The transformation matrices are of the form

$$\mathbf{H} = \begin{bmatrix} & & & x \\ & \mathbf{R} & & y \\ & & & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{205}$$

where x, y, and z represent the translation from one end of the element to another and $\mathbf{R}$ represents rotation of the distal links (gross body rotations of the equilibrium position). $\mathbf{R}$ is normally used only by joints in two and three dimensional models. $\mathbf{R}$ is one of

$$\mathbf{R}_{\mathrm{x}} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{bmatrix} \tag{206}$$

$$\mathbf{R}_{\mathrm{y}} = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix} \tag{207}$$

$$\mathbf{R}_{\mathrm{z}} = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{208}$$

or for an element with no rotation

$$\mathbf{R}_{\mathrm{I}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{209}$$

Note that these rotation matrices are used to define the nominal position of the robot. The TMM models the linear response of the system about this nominal position. θ_x, θ_y, and θ_z can be large angles, but they are constants.

Beam and rigid link elements have transformation matrices of the form

$$\mathbf{H}_{\mathrm{L}} = \begin{bmatrix} 1 & 0 & 0 & L \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{210}$$

and torsional springs and many actuators use

$$\mathbf{H}_{\mathrm{I}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{211}$$

The need for the user to define GetHT for most new elements can be eliminated by defining two helper classes:

```
1 class TMMElementLHT(TMMElement):
2     def GetHT(self):
```

```
3           return HT4(x=self.params['L'])
4
5   class TMMElementIHT(TMMElement):
6       def GetHT(self):
7           return HT4()
```

where TMMElementLHT returns a homogeneous transformation matrix like $\mathbf{H}_{\mathrm{L}}$ in equation (210). Beam and rigid link elements would now derive from this class. TMMElementIHT returns a homogeneous transformation matrix like $\mathbf{H}_{\mathrm{I}}$ in equation (211). Spring/damper and actuator elements would derive from it. TMMElementLHT and TMMElementIHT are examples of the ease and convenience of inheritance in Python. They are each only 3 lines of code, but they specialize the TMMElement class in a useful way by deriving from it. The fact that they derive from TMMElement is captured by having the base class name in parentheses (TMMElement) in lines 1 and 5. Each of them can become the base class for further specialization by other classes.

The HT4 function called by TMMElementLHT and TMMElementIHT is

```
1   def HT4(axis='',angle=0,x=0.,y=0.,z=0.):
2       #4x4 homogeneous transformation matrix
3       if axis:
4           r1=rot3by3(axis,angle)
5       else:
6           r1=scipy.eye(3,dtype='d')
7       s1=c_[r1,array([[x],[y],[z]])]
8       return r_[s1,array([[0.,0.,0.,1.]])]
```

which in turn calls

```
1   def rot3by3(axis,angle):
2       rad=angle*pi/180.
3       if isinstance(axis,str):
4           if axis.lower()=='x':
5               axis=1
6           elif axis.lower()=='y':
7               axis=2
8           elif axis.lower()=='z':
9               axis=3
10      if axis==1:
11          R=array([[1.0,0.0,0.0],[0.0,cos(rad),-sin(rad)],[0.0,sin(rad),
                cos(rad)]])
```

```
12      elif axis==2:
13          R=array([[cos(rad),0.0,sin(rad)],[0.0,1.0,0.0],[-sin(rad),0.0,
                cos(rad)]])
14      elif axis==3:
15          R=array([[cos(rad),-sin(rad),0.0],[sin(rad),cos(rad)
                ,0.0],[0.0,0.0,1.0]])
16      return R
```

Lines 3–9 convert axis to an integer if the user inputs 'x', 'y', or 'z'. Lines 10–15 are an **if** block returning either $\mathbf{R}_x$, $\mathbf{R}_y$, or $\mathbf{R}_z$ from equations (206)–(208) depending on axis.

If users derive from either TMMElementLHT or TMMElementIHT, they only need to define GetMat. Provided that GetMat is compatible with a new symbolic string class, all of the symbolic parts of the analysis will be taken care of automatically. This requires that GetMat be written in such a way that it can return either a numeric or a symbolic transfer matrix. This will be discussed in greater detail in section 6.9.

6.5.5 Example: Torsional Spring/Damper

As an example of how to derive a new type of transfer matrix element from the TMMElement class, consider a torsional spring/damper.

```
1  class TorsionalSpringDamper(TMMIHTElement):
2      def GetMat(self,s,sym=False):
3          N=self.maxsize
4          if sym:
5              myparams=self.symparams
6          else:
7              myparams=self.params
8          k=myparams['k']
9          c=myparams['c']
10         springterm=1/(k[0]+c[0]*s)
11         if sym:
12             maxlen=len(springterm)+10
13             matout=eye(N,dtype='f')
14             matout=matout.astype('S%d'%maxlen)
15         else:
16             matout=eye(N,dtype='D')
17         matout[1,2]=springterm
18         if max(shape(k))>1 and self.maxsize>=8:
19             matout[5,6]=1/(k[1]+c[1]*s)
20         if max(shape(k))>2 and self.maxsize>=12:
```

```
21                matout[9,10]=1/(k[2]+c[2]*s)
22            return matout
```

This is all the code that is need to fully define a TorsionalSpringDamper element. Line 1 specifies that the class TorsionalSpringDamper inherits from TMMElementIHT (i.e. its homogeneous transformation matrix is an identity matrix). Lines 2–22 override the GetMat method. Lines 4–7 are an **if** block. Either numeric or symbolic values for the parameters are retrieved based on whether or not symbolic analysis is being performed. Lines 11–16 are a similar **if** block, creating an identity matrix that is either numeric or symbolic. Line 17 substitutes the spring term $1/(k + cs)$ into the appropriate spot in the matrix. Lines 18–21 make the same substitution for the other axes if this spring is part of two or three dimension TMM analysis. The 1D case would take the form:

$$\mathbf{U}_{\text{spring/damper}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & \frac{1}{c\,s+k} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{212}$$

This is an example of how user extensibility works. The user can extend the TMM analysis software by defining a new element. This can be done simply by overriding the one method GetMat for most elements (or GetAugMat for actuators).

Once this new class has been defined, it would actually be used like this

```
1   from TMM.spring import TorsionalSpringDamper
2   myspring = TorsionalSpringDamper({'k':10,'c':5},maxsize=4)
3   U = myspring.GetMat(1.0j)
```

which gives

$$\mathbf{U} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0.08 - 0.04\text{j} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{213}$$

Line 1 is the Python code necessary to load a class that is defined in a file. Line 2 calls the __init__ method of the TorsionalSpringDamper class. maxsize=4 specifies that 1D transfer matrix analysis is being performed (the transfer matrix for the element will be 4×4).

For the symbolic case:

```
s=symstr('s')
U=myspring.GetMat(s,sym=True)
```

produces

$$\mathbf{U} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1/(k+c\,s) & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{214}$$

For the sake of modeling SAMII, this work included deriving transfer matrix elements for beams, rigid links, and angular velocity sources along with using torsional spring/damper elements to model unactuated joints and localized compliance.

6.6 Design of the TMMSystem Class

6.6.1 Purpose/Overview

The TMMElement class is the primary building block of the analysis package and deriving new elements from it is the primary means of user extensibility. Once a TMMElement class has been derived for each element type in the system, those elements must be grouped into a TMMSystem model. The TMMSystem class will be the work horse for much of the analysis and calculations associated with the transfer matrix method.

A TMMSystem is an object and users can derive from it if they need to. One simple example of how this might be useful is deriving a new class from TMMSystem with certain boundary conditions. If the user is working primarily on serial robots and they are typically clamped at the base and free and the tip, a new ClampedFreeTMMSystem could be derived with default boundary conditions:

```
class ClampedFreeTMMSystem(TMM.TMMSystem):
    def __init__(self,elemlist,bcend='free',bcbase='fixed',bodeouts=[]):
        TMM.TMMSystem.__init__(self,elemlist,bcend=bcend,bcbase=bcbase,
            bodeouts=bodeouts)
```

Note that this class only takes up three lines of code. Creating this class is easy to do and would free the user from thinking about this coding detail. It saves the user from having

to remember or look up the convention for whether the tip or base boundary condition is first.

6.6.2 Methods

The primary functions of a TMMSystem are finding the natural frequencies and mode shapes of the system as well as the Bode responses to any inputs. In order to find the natural frequencies and mode shapes, there will need to be methods for finding the system transfer matrix (based on the transfer matrices of the elements) as well as the characteristic determinant of the system and then searching numerically for values of s that drive the characteristic determinant to 0. It will be advantageous to do much of this analysis both numerically and symbolically.

The TMMSystem methods are by far the lengthiest portion of code associated with this thesis, with nearly 1700 lines of code. There are large chunks dedicated to symbolic analysis and integrated system identification. Symbolic analysis will be talked about in greater detail in section 6.9 and integrated system identification will be discussed in chapter 7.

6.6.3 Properties

A TMMSystem really has only three properties: a list of TMMElement instances (with their parameters specified), the system boundary conditions, and a list of Bode outputs that are of interest to the modeler. A TMMSystem instance could be made up of TMMSubSystem's instead of TMMElement's to make it easy to reuse parts of the model. This could be helpful if a user wants to change a model from open to closed-loop by switching only the actuator model and keeping the rest of the structural model the same.

6.6.4 Code

Here is an example slice of TMMSystem code that shows how the system transfer matrix and augmented transfer matrix are calculated. This code is made particularly simple by the object-oriented nature of the analysis package. Each TMMElement must have GetMat and GetAugMat methods. The TMMSystem code can call that method for each element in order to multiply them together to determine the system transfer matrix.

```
1  class TMMSystem:
2      ...
3      def FindU(self,value):
4          if scipy.shape(value):
5              value=value[0]+value[1]*1j
6          U=scipy.eye(self.maxsize,dtype='D')
7          for curelem in self.elemlist:
8              tempU=curelem.GetMat(value)
9              U=scipy.matrixmultiply(tempU,U)
10         return U
11
12     def FindAugU(self,value):
13         if scipy.shape(value):
14             value=value[0]+value[1]*1j
15         U=scipy.eye(self.maxsize+1,dtype='D')
16         for curelem in self.elemlist:
17             tempU=curelem.GetAugMat(value)
18             U=scipy.matrixmultiply(tempU,U)
19         return U
20     ...
```

Lines 3–10 define the FindU method which calculates the system transfer matrix at a certain value of s. Lines 4 and 5 are an **if** block to handle the possibility that value is an array of [real(s),imag(s)] rather than a complex number. Line 6 initializes the system transfer matrix with an identity matrix. (dtype='D' specifies that the matrix is made up of complex numbers.) Lines 7–9 are a for loop that iterates over each element in the TMMSystem model. Line 8 gets the transfer matrix for the current element and line 9 multiplies the element matrix by the system matrix up to that point in the model.

Lines 12–19 define the function FindAugU which finds the augmented system transfer. The code is very similar to that for FindU, except that GetAugMat is called for each element and the matrices are defined to be N+1 by N+1.

6.6.5 An Example

As an example of using the TMMSystem class to find the natural frequencies of a system, consider a system composed of only one element – a cantilever beam. Before searching for the natural frequencies numerically, decent initial guesses are needed. To that end, the

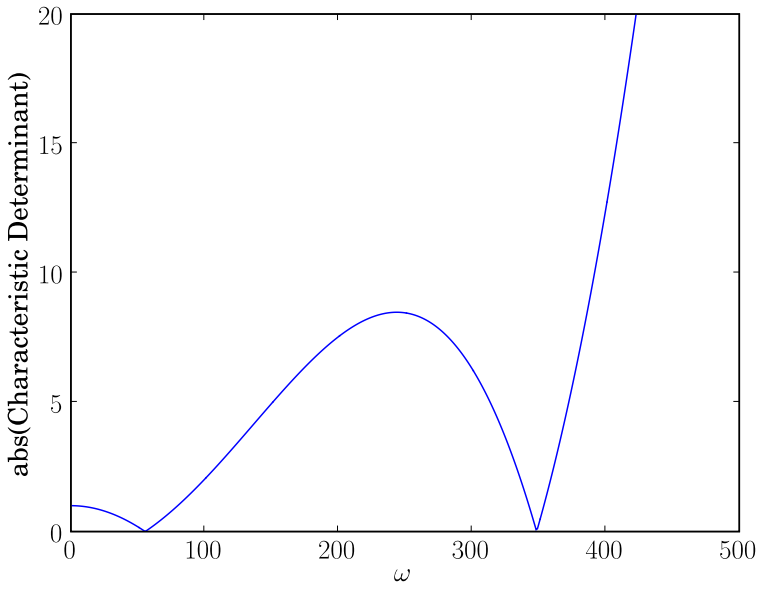

Figure 105: The absolute value of the characteristic determinant of a cantilever beam over a frequency range that includes the first two natural frequencies.

characteristic determinant is plotted over a range of frequencies ω:

```
1 from TMM.TMMSystem import TMMSystem
2 from TMM.beam import BeamElement
3 mybeam=BeamElement({'EI':50.0,'mu':0.2,'L':1},maxsize=4)
4 mysystem=TMMSystem([mybeam],'free','fixed')
5 ovect=arange(1,500,1)
6 svect=1.0j*ovect
7 cdlist=[mysystem.EigError(s) for s in svect]
8 cdvect=array(cdlist)
9 plot(ovect,cdvect)
```

Lines 1 and 2 **import** the TMMSystem and BeamElement classes. Line 3 initializes the beam element. Line 4 initializes the TMMSystem. Line 5 creates a list of ω values over which the characteristic determinant will be plotted. Line 6 converts those ω values to s values. Line 7 calculates the characteristic determinant at each point in the list of s values using the EigError method. The syntax of this line makes use of list comprehension to create an implied **for** loop. Line 8 converts the list of characteristic determinant values to an array. Line 9 plots the characteristic determinant array vs. ω, as shown in Figure 105.

From Figure 105, it is apparent that $\omega = 50$ and 350 are good initial guesses for the numerical search. Using these values to initiate the search

```
1 s1=mysystem.FindEig(50.0j)
2 s2=mysystem.FindEig(350.0j)
```

gives $s_1 = 55.593083j$ and $s_2 = 348.395902j$. From theoretical analysis, it is known that the natural frequencies of a cantilever beam can be found by first solving for β such that

$$\cos\beta \ \cosh\beta + 1 = 0 \tag{215}$$

and then finding the natural frequencies from

$$\omega_i = \beta_i^2 \sqrt{\frac{EI}{\mu L^4}} \tag{216}$$

Solving the problem using this theoretical analysis can be done using the following Python code:

```
1  EI=50.0
2  mu=0.2
3  L=1.0
4  def myfunc(x):
5      return cos(x)*cosh(x)+1
6
7  b1=optimize.newton(myfunc,1.8)
8  b2=optimize.newton(myfunc,4.6)
9  w1=b1**2*sqrt(EI/(mu*L**4))
10 w2=b2**2*sqrt(EI/(mu*L**4))
```

Lines 4 and 5 define the function whose roots are sought (equation (215)). Lines 7 and 8 use this function along with Newton's method to find the first two values for β that satisfy equation (215) yielding $\beta_1 = 1.875104$, $\beta_2 = 4.694091$. Lines 9 and 10 convert these β values to ω's giving $\omega_1 = 55.593083$, and $\omega_2 = 348.395902$. These are the same values found by the TMM analysis, validating the TMMSystem code.

While this example has focused on a very simple system, the analysis steps do not change for a more complicated system. All that would need to change is that the user pass a longer list of elements to the code that creates the system model, i.e. replace

```
1 mysystem=TMMSystem([mybeam],'fixed','free')
```

with

```
mysystem=TMMSystem([element1, element2, element3, ...],'fixed','free')
```

6.7 Bodeout Class

The bodeout class is used to describe the outputs that should be calculated when performing Bode analysis using the TMM (either numeric or symbolic Bode analysis). It basically provides a clean way to encapsulate all of the information needed to simulate a measured signal from a TMM model. This includes the location in the model where the measurement is to be taken, whether the measurement is absolute or relative to another position in the model, any gain associated with the sensor, and whether or not there is any post-processing that needs to be done on the signal. Currently, post-processing is limited to calculating velocity or acceleration from position signals. It also provides the option of specifying seedfreq and seedphase which are used to unwrap the phase portion of Bode outputs.

The bodeout class takes up very little code:

```
class bodeout:
    """This class is used by transfer matrix models
    to define a desired bode output."""
    def __init__(self, output='',input='',ind=None,pind=None,type='abs',
        post='',dof=None,gain=1.0,gainknown=True,seedfreq=-1,seedphase
        =0.0):
         self.output=output
         self.input=input
         self.ind=ind
         self.pind=pind
         self.dof=dof
         self.post=post
         self.type=type
         self.gain=gain
         self.gainknown=gainknown
         self.seedfreq=seedfreq
         self.seedphase=seedphase
```

The bodeout class has an __init__ method only. This allows users to create instances of the class, but those classes do not actually do anything (they have no other methods besides the __init__ method needed to create the instances). The class serves only as a means of

encapsulating all of the parameters needed to calculate Bode outputs. All of the calculations are done on it by the TMMSystem class.

6.8 TMMSubSystem Class

The TMMSubSystem class serves two purposes. First it provides a clean and easy way to group chunks of a TMMSystem so that some parts can be changed while others are reused. For example, in modeling SAMII three different system models are used with only one or two elements changing from one model to another. An open-loop actuator model is used for system identification. The actuator is replaced with a θ feedback model and an acceleration feedback element is added later to model the closing of two nested feedback loops. Not only does the TMMSubSystem class make it easier to create these three models, but it prevents errors that might be made in creating the three separate models from scratch.

Consider the mathematical representations of the three TMM system models for SAMII. Open loop:

$$\mathbf{z}_{\text{tip}} = \mathbf{U}_{\text{link2}} \boxed{\mathbf{U}_{\text{ol}}} \mathbf{U}_{\text{link1}} \mathbf{U}_{\text{joint1}} \mathbf{U}_{\text{link0}} \mathbf{U}_{\text{beam}} \mathbf{U}_{\text{basespring}} \mathbf{z}_{\text{base}} \tag{217}$$

With θ feedback:

$$\mathbf{z}_{\text{tip}} = \mathbf{U}_{\text{link2}} \boxed{\mathbf{U}_{\text{cl}}} \mathbf{U}_{\text{link1}} \mathbf{U}_{\text{joint1}} \mathbf{U}_{\text{link0}} \mathbf{U}_{\text{beam}} \mathbf{U}_{\text{basespring}} \mathbf{z}_{\text{base}} \tag{218}$$

With vibration suppression:

$$\mathbf{z}_{\text{tip}} = \mathbf{U}_{\text{link2}} \boxed{\mathbf{U}_{\text{cl}} \mathbf{U}_{\text{acc}}} \mathbf{U}_{\text{link1}} \mathbf{U}_{\text{joint1}} \mathbf{U}_{\text{link0}} \mathbf{U}_{\text{beam}} \mathbf{U}_{\text{basespring}} \mathbf{z}_{\text{base}} \tag{219}$$

If a TMMSubSystem is defined whose transfer matrix represents

$$\mathbf{U}_{\text{subsys}} = \mathbf{U}_{\text{link1}} \mathbf{U}_{\text{joint1}} \mathbf{U}_{\text{link0}} \mathbf{U}_{\text{beam}} \mathbf{U}_{\text{basespring}} \tag{220}$$

then these same three models can be represented by

$$\text{Open loop: } \mathbf{z}_{\text{tip}} = \mathbf{U}_{\text{link2}} \mathbf{U}_{\text{ol}} \mathbf{U}_{\text{subsys}} \mathbf{z}_{\text{base}} \tag{221}$$

$$\theta \text{ feedback: } \mathbf{z}_{\text{tip}} = \mathbf{U}_{\text{link2}} \mathbf{U}_{\text{cl}} \mathbf{U}_{\text{subsys}} \mathbf{z}_{\text{base}} \tag{222}$$

$$\text{With vibration suppression: } \mathbf{z}_{\text{tip}} = \mathbf{U}_{\text{link2}} \mathbf{U}_{\text{cl}} \mathbf{U}_{\text{acc}} \mathbf{U}_{\text{subsys}} \mathbf{z}_{\text{base}} \tag{223}$$

The TMMSubSystem class provides a convenient way to reuse model portions in this way.

The second purpose of the TMMSubSystem class is to provide a convenient way to model portions of a TMMSystem that are used in non-collocated feedback, as discussed in section 3.3. The transfer matrix of a portion of the TMMSystem needs to be inverted to find the non-collocated states as shown in equation (86) (repeated here):

$$\mathbf{z}_{\text{beam}} = \left(\mathbf{U}_{\text{link1}}\,\mathbf{U}_{\text{joint1}}\,\mathbf{U}_{\text{link0}}\right)^{-1}\,\mathbf{z}_{\text{before}} \tag{224}$$

The TMMSubSystem class provides a way to identifying such a portion of a TMMSystem model, and could be used in automating the generation of non-collocated feedback elements.

6.9 Software Design for Symbolic Modeling

The goal of software design for symbolic modeling is to make the symbolic capabilities of the analysis package available with very little additional effort on the part of the user. This is accomplished by inheriting all of the necessary symbolic methods from TMMElement when deriving a new element class, provided that the GetMat method for the new element is written to be compatible with a symbolic string class that was created for this purpose.

Here is an example of a GetMat function from a TorsionalSpringDamper element that can return either a numeric or symbolic string matrix:

```
1  def GetMat(self,s,sym=False):
2      N=self.maxsize
3      if sym:
4          myparams=self.symparams
5      else:
6          myparams=self.params
7      k=myparams['k']
8      c=myparams['c']
9      springterm=1/(k[0]+c[0]*s)
10     if sym:
11         maxlen=len(springterm)+10
12         matout=eye(N,dtype='f')
13         matout=matout.astype('S%d'%maxlen)
14     else:
15         matout=eye(N,dtype='D')
16     matout[1,2]=springterm
17     if max(shape(k))>1 and self.maxsize>=8:
```

```
18            matout[5,6]=1/(k[1]+c[1]*s)
19        if max(shape(k))>2 and self.maxsize>=12:
20            matout[9,10]=1/(k[2]+c[2]*s)
21        return matout
```

There are two places were the code checks to see if a symbolic matrix is requested. Lines 3–6 retrieve two different sets of parameters for myparams depending on whether or not symbolic analysis has been requested. params contains numeric values for the coefficients while symparams contains symbolic strings. Lines 10–15 determine whether the output matrix should be a numeric representation or a string representation.

It can be a small hassle to the authors of new TMM element classes to have to make sure that their GetMat function is compatible with symbolic strings, but this is much easier than the alternative which is to require them to write both a GetMat function and a GetSymMat function. GetSymMat is used to generate Maxima code for symbolic analysis. Writing such a function would require users to learn at least a little Maxima (this option is left open if the user wants to override the inherited GetSymMat function). Another advantage of this polymorphic approach that can handle either numeric or symbolic string inputs and outputs is that it provides a nice way to check the GetMat function. The symbolic string version of the output can easily be turned into LaTeX code. Running LaTeX then produces a typeset representation of the matrix which is much easier to read and debug than other code.

The key to being able to use the same code for numeric or symbolic output is the creation of a symbolic string class. Python allows operator overloading so that the meaning of ordinary math operations can be defined for this new class. For example, the code `'a'+'b'` should produce `'a+b'`. And if `p1='a+b'` and `p2='c+d'`, then `p1*p2` needs to return `'(a+b)*(c+d)'`. While there have been several other attempts to add basic symbolic capabilities to Python, none of them meet the needs of this problem exactly. They are either unnecessarily complicated or not completely developed. This work does not require the development of a full set of symbolic tools in Python, only enough to generate a valid input script to Maxima, where the bulk of the symbolic analysis is done. A symstr class has been created to meet this need. The symstr class is basically a string class that overrides mathematical operators like

```
1 def __add__(self,other):
2     return symstr(self.__str__()+'+'+str(other))
```

This function overrides the definition of the '+' operator for a symbolic string. Symbolic strings can then be added together like this

```
1 a=symstr('a')
2 b=symstr('b')
3 a+b
```

producing `'a+b'`.

Combining the symstr class with polymorphic trigonometric and hyperbolic functions that can take either floating point or symstr inputs has been sufficient to create polymorphic GetMat functions for all the elements used to model SAMII. Here is an example of a polymorphic cosine function:

```
1 def cos(ent):
2     if isinstance(ent,str):
3         return symstr('cos('+ent+')')
4     else:
5         return scipy.cos(ent)
```

Lines 2 and 3 check to see if the input is a string and return `'cos(x)'` if x is a string. Lines 4 and 5 return the numeric value of cos(x) if x is not a string.

Once the user has written a symbolic string compatible GetMat function (or GetAugMat if the element is an actuator), the new TMMElement can be used in symbolic analysis. The new element will inherit from the base class everything else needed to perform symbolic analysis. The symparams property used in GetMat will automatically be created based on the names of the parameters in params (provided that the __init__ method of the new element calls TMMElement.__init__ at some point). The other methods that are needed are inherited from the base class: GetSymMat, GetAugSymMat, and GetMaximaLines.

CHAPTER VII

SYSTEM IDENTIFICATION

7.1 Introduction

This chapter first discusses the integration of system identification capabilities into the software. The clean integration of system identification capabilities into the software combined with intelligent automation of the identification process greatly eases what can otherwise be a time consuming process with the potential for a lot of low-level errors.

The second part of this chapter presents the system identification approach used for SAMII as a case study. The use of the software is demonstrated and the need for these capabilities is clarified.

7.2 The Need

The primary focus of system identification for SAMII is to determine the values of unknown system parameters like joint stiffness coefficients that will lead to quantitative agreement between the TMM model and experimental results, i.e. Bode plots. Existing system identification software cannot be used for this purpose, because the software does not have TMM modeling capabilities. Existing software is focused on fitting a discretized model to the experimental data. Such a model is not very useful in determining the parameters of the TMM model. This chapter presents an integrated approach to system ID for determining the parameters of a TMM model.

7.3 Motivation

Beyond making the TMM accessible to controls engineers, an additional goal of the software design is to make all aspects of modeling and control design for flexible structures easier and faster so that the controls or vibrations engineer spends less time and effort on low-level computing tasks and gets more consistent results. One way this is done is through

integrating common control design tasks directly into the software. An example of this is automating the system identification process.

System identification can be a tedious and error prone process with lots of low-level programming. Ordinarily, a controls engineer would need to hand code a script that calls an optimization routine. The inputs to these optimization routines typically include a vector of unknown coefficients and the name of a cost function. The cost function must take the unknown coefficient vector and return a scalar value that is to be minimized. Typically this would be a squared sum of the error between model and experiment, possibly with some weighting vector. The cost function must some how assign the parameters in the coefficient vector to the correct system variables. This may involve creating a new model each time with the current coefficients. The cost function will also need to compare the model output with the experimental data to get the error value that is to be minimized.

This software package seeks to handle these low-level details automatically and free the user to focus on higher-level tasks.

7.4 Overview

In formulating a transfer matrix model of a system, some parameters are likely to be unknown. For SAMII, the unknown parameters are the joint stiffness and damping coefficients. In an overly simple model of the system, the joints might be considered rigid. However, experimental results show that joint compliance is significant. If a quantitatively accurate model of the system is sought, these coefficients must be determined. This can be done by finding the coefficients that minimize the error between experimental Bode plots and those generated by a model. This minimization will require setting up an appropriate cost function and then passing that cost function to an optimization routine. This can be a labor intensive and error prone process. Integrating system identification into the TMM analysis software can reduce the amount of low-level thinking the controls engineer must do and reduce or eliminate common programming errors.

System ID will be done in an object-oriented fashion by allowing the user to specify for each element in the model which parameters are unknown. For SAMII, the parameters

of the beam and rigid mass elements are known from material proprieties and physical dimensions while the stiffness and damping coefficients of the joints and the gain and first-order lag of the actuator are unknown. Based on these element-level specifications, a vector of all unknown parameters will be generated and a Bode function will be automatically created that takes this vector of unknown parameters as an input. This function will come from symbolic TMM analysis. The error weights of each Bode magnitude and phase can be specified when the Bode outputs for the TMM system are created. These weights can be used to automatically create a cost function that calls the automatically created Bode function.

The required user inputs for system identification are

- Specify a TMM model
- Indicate unknown parameters for each element
- Specify the constant relative weights of the errors for each Bode output (magnitude and phase)
- Give the location of the experimental data files

The system identification function will then work through these steps automatically:

- Generate a Bode function based on symbolic analysis that takes a vector of unknown parameters as an input
- Create a cost function
- Generate a script that runs the optimization (this script will be user editable for further customization)

7.5 Goal

The goal should be to intelligently automate this process, minimizing the required user input and keeping that input on a high level. Ideally, it should not take too much more code to set up the system identification routine than it would take in words to say "I want to make this model, with these parameters unknown, and fit it to this set of data."

Not only does the software need to automate and correctly handle all of the low-level details associated with system identification, but it must also be flexible enough to allow intelligent approaches to system ID and it must incorporate the practical needs and concerns of identifying real systems.

Practical needs include things like automated data processing, data truncation, and down-sampling.

The software design goals could be summarized in the following way:

- Need software that is flexible enough to support intelligent system ID approaches
- It must be easy for the user to specify the model and which coefficients are unknown
- The software must do the low-level tasks automatically, such as generating scripts that define the symbolic Bode model and the cost function

7.6 Intelligent Automation

Figure 106 illustrates the need for intelligent automation. While this work seeks to automate many common controls engineering tasks through integrating these tasks into the software, this automation must be done carefully and with ways for humans to check the results in appropriate places. The Bode plot shown was generated as an intermediate step during the development of integrated system identification capabilities. The data being curve fit was saved with phase wrapping taken care of while the Bode response being generated by the model shows a phase jump at 10 Hz that should be corrected (the phase as shown jumps from –180° to +180° at 10 Hz because of the definition of the atan2 function.). If this plot had not been generated and checked, the curve fitting results might have been very strange and the cause might have been hard to track down. The term intelligent automation refers to the fact that that experience and intellect guide the programmer in building in these kind of checks. The user must be convinced that the software has correctly automated the process and the results are trustworthy.

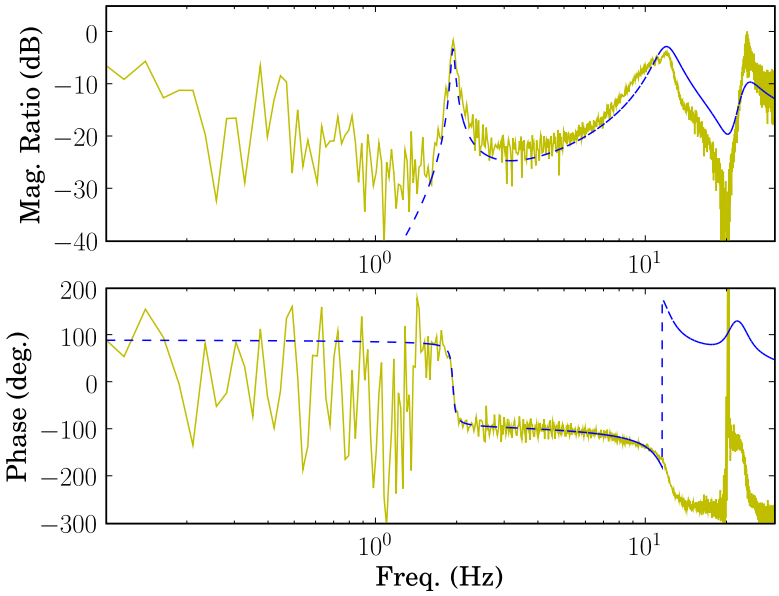

Figure 106: A phase wrapping problem that needs to be corrected before system identification is attempted.

7.7 Software Implementation

The basic approach to be followed in implementing this idea is to combined the symbolic modeling capabilities already built into the software with the ability to specify unknown parameters in the model. Each TMMElement has a property called unknownparams which will contain a list of the coefficients for that element that are to be treated as unknown when the symbolic Bode response for the system is being determined. The specification of one of the unknown spring/damper elements of SAMII would look like this:

```
basespring = TorsionalSpringDamper4x4({'k':166358.0,'c':468.789},
    symlabel = 'base',unknownparams = ['k','c'])
```

The symbolic analysis code can automatically generate Bode plots and cost functions that take a vector of unknown coefficients and assign them to the correct parameters. This eliminates the need to recreate a new model with each new set of coefficients as there would be without the symbolic modeling capabilities. System identification is one area where the speed benefits of symbolic analysis really pay off. The optimization routine used to perform the system ID will call the underlying Bode functions many times. It is much faster to

evaluate a closed-form expression for the Bode function than to repeat the numeric matrix calculations each time. For SAMII's system ID, running the optimization with the same settings for maximum iterations, the symbolic version is approximately 80 time faster than the purely numeric version.

The bodeout class plays a key role in system identification. It makes comparison between predicted and experimental Bode plots very clean and provides a means of specifying the phase unwrapping hints and all other information needed for cost-function generation.

7.7.1 An Example

An entire model for SAMII specifying all the unknowns takes only 12 lines of code:

```
1  def olsamiimodel_withig():
2      basespring=TorsionalSpringDamper4x4({'k':166358.0,'c':468.789},
           symlabel='base',unknownparams=['k','c'])
3      beam=samiiBeam()
4      link0=samiiLink0()
5      j1spring=TorsionalSpringDamper4x4({'k':4028.28,'c': 6.3058},symlabel
           ='j1',unknownparams=['k','c'])
6      link1=samiiLink1()
7      avs=AngularVelocitySource4x4({'K':0.435489,'tau':173.833},symlabel='
           act',unknownparams=['K','tau'])
8      j2spring=TorsionalSpringDamper4x4({'k':1900.49,'c':21.6805},symlabel
           ='j2',unknownparams='all')
9      bodeout1=bodeout(input='j2v', output='a1', type='abs', ind=beam,
           post='accel', dof=0, gain=0.35, gainknown=False)
10     bodeout2=bodeout(input='j2v', output='j2a', type='diff', ind=[
           j2spring,link1], post='', dof=1, gain=180.0/pi)
11     link2=samiiLink2()
12     return ClampedFreeTMMSystem([basespring,beam,link0,j1spring,link1,
           avs,j2spring,link2],bodeouts=[bodeout1,bodeout2])
```

It only takes three lines of code to import and initialize this model and use it to set up the system ID:

```
1  import curvefit_samii_model as cfsm
2  olmodel=cfsm.olsamiimodel_withig()
3  olmodel.IntegratedCurveFit(curvefitdata,'example_curvefit')
```

These three lines of code are an example of powerful analysis capabilities being accessible in a only a few lines of code once a TMMSystem model has been created.

IntegratedCurveFit takes two inputs: the name of a module containing the experimental data and a label to associate with the Bode and cost functions that it generates. Calling IntegratedCurveFit automatically generates the vector of initial guesses based on the unknown parameters specified in olsamiimodel_withig. It also creates symbolic Bode functions for all bodeouts that take the vector of unknown coefficients as an input (these symbolic Bode functions are generated in FORTRAN and Python). It also compiles the FORTRAN files and automatically generates a script that contains the cost function for the curve-fitting.

The bodeout class makes the creation of the cost function very straightforward. The cost function iterates over all of the defined bodeout's and calculates the sum of the squared error between experimental and model Bode responses for each output. Here is an example of an automatically generated cost function:

```
1  def mycost(ucv,phaseweight=0.1,returnbl=False):
2          totale=0.0
3          if returnbl:
4                  bl=[]
5          for cf,cb in zip(funclist,mybodes):
6                  curc=cf(s,ucv)
7                  curb=rwkbode.rwkbode(cb.output,cb.input,compin=curc)
8                  curb.seedfreq=cb.seedfreq
9                  curb.seedphase=cb.seedphase
10                 curb.PhaseMassage(fitf)
11                 magE=squeeze(cb.dBmag())-squeeze(curb.dBmag())
12                 phaseE=squeeze(cb.phase)-squeeze(curb.phase)
13                 totale+=sum(magE**2)+phaseweight*sum(phaseE**2)
14                 if returnbl:
15                         bl.append(curb)
16         if returnbl:
17                 return totale,bl
18         else:
19                 return totale
```

The cost function has one required input, a vector of unknown coefficients ucv. There are two optional inputs, the weighting factor to be applied to phase error phaseweight and a flag specifying whether or not the function should output the model Bode list. Lines 5–15 are a **for** loop that iterates over all of the Bode outputs and calculates the error for each one. Lines 16–19 return either the total error or the total error and the Bode list, depending on

the value of returnbl.

The optimization can be run by executing the script containing the cost function:

```
run example_curvefit_autofitfuncf.py
```

7.7.2 Automated Data Processing

System ID can also involve significant amounts of experimental data that needs to be processed and analyzed in a consistent and efficient manner. Classes have been created that represent experimental Bode tests and reports. These classes form the basis of automated data processing.

7.8 SAMII System ID Case Study

7.8.1 Overview

A practical example of system identification on SAMII will help clarify the need for semi-automated system identification with clean integration into the modeling software. This section will make two primary points about software design and system identification. First, system ID will often involve several different models and possibly several different sets of data. The software should automate the process and handle the low-level details so that the controls or vibrations engineer can easily perform the analysis as many times as necessary and get consistent results. The second point is that the automated process must be flexible enough to allow for creative and intelligent approaches to system identification.

7.8.2 System Description

Initial work on SAMII has focused on two outputs: the angular position of joint 2 (θ) and the acceleration at the end of the cantilever beam ($\ddot{x}$). The input to the system is the command voltage to joint 2 (v). θ and $\ddot{x}$ were shown on a picture of SAMII in Figure 20. Figures 107 and 108 show the Bode plots for the experimental data. In all tests in this chapter, the system is excited by a swept sine input to joint 2 with an amplitude of 1.5°. All joints are under feedback control. The desired angle is held constant for all joints besides joint 2, so that SAMII is held in a nominal position. The real-time control and data acquisition system is running at 500 Hz.

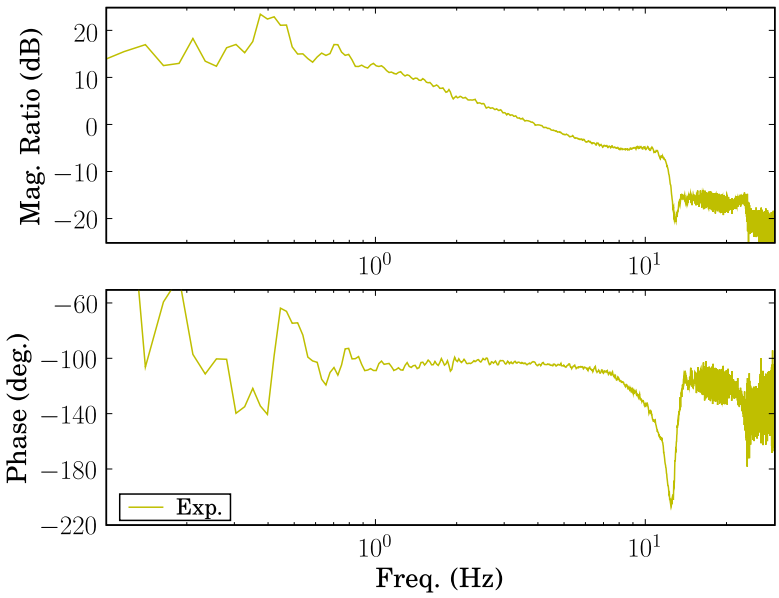

Figure 107: Experimental actuator Bode plot θ/v.

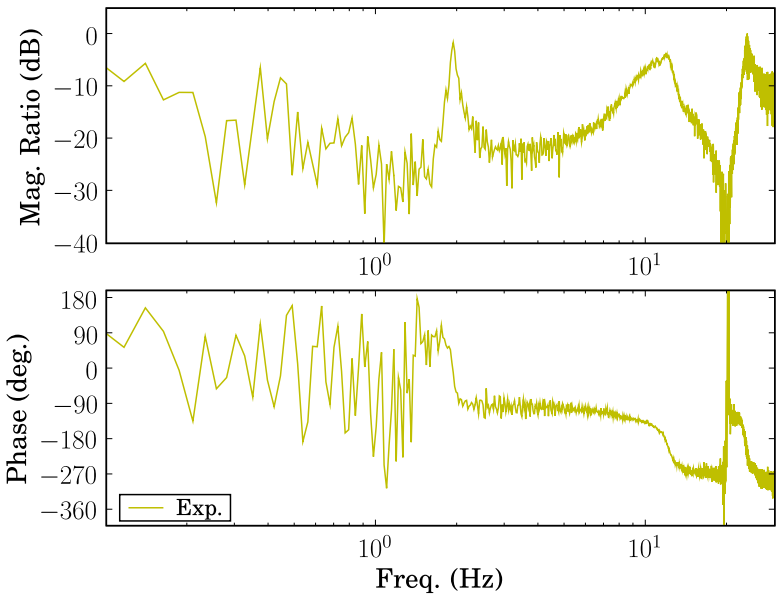

Figure 108: Experimental flexible base Bode plot $\ddot{x}/v$.

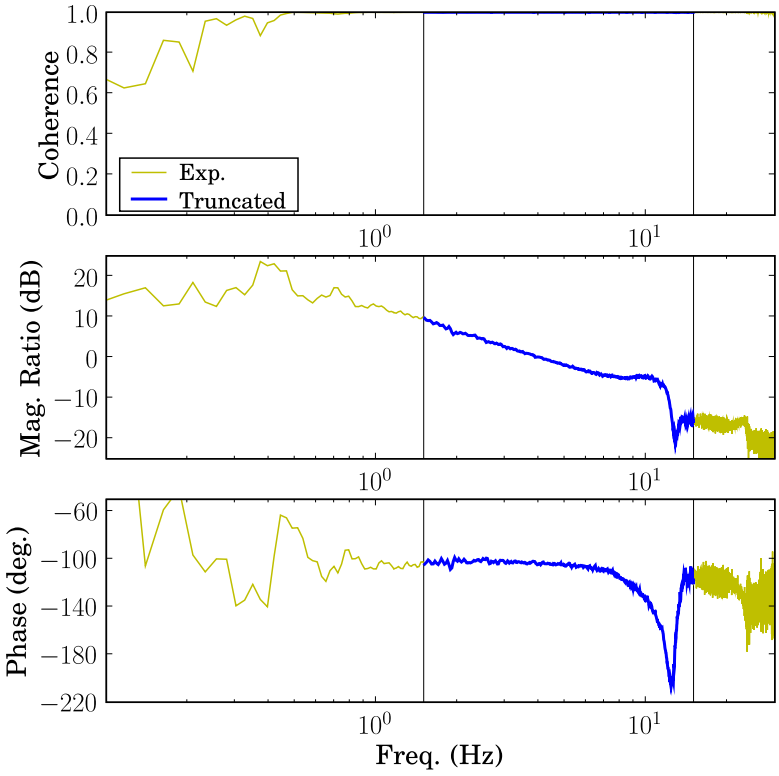

Figure 109: Actuator Bode plot θ/v with coherence showing the truncated range for initial system identification.

7.8.3 Data Truncation

The first problem to overcome is to figure out how much of this experimental data to curve fit in the system identification routine. The modeler is most likely concerned with the response of the system over a certain frequency range. Trying to fit too large a frequency range may degrade the model's accuracy within the frequency range of interest. This will be especially true if actuator limitations result in the system not being excited at all frequencies. This question can be answered based on the coherence plots. Figures 109 and 110 combined the Bode plots with coherence. Especially for Figure 110, the coherence seems to be good from approximately 1.5–15 Hz. This range also includes the first two modes of the system.

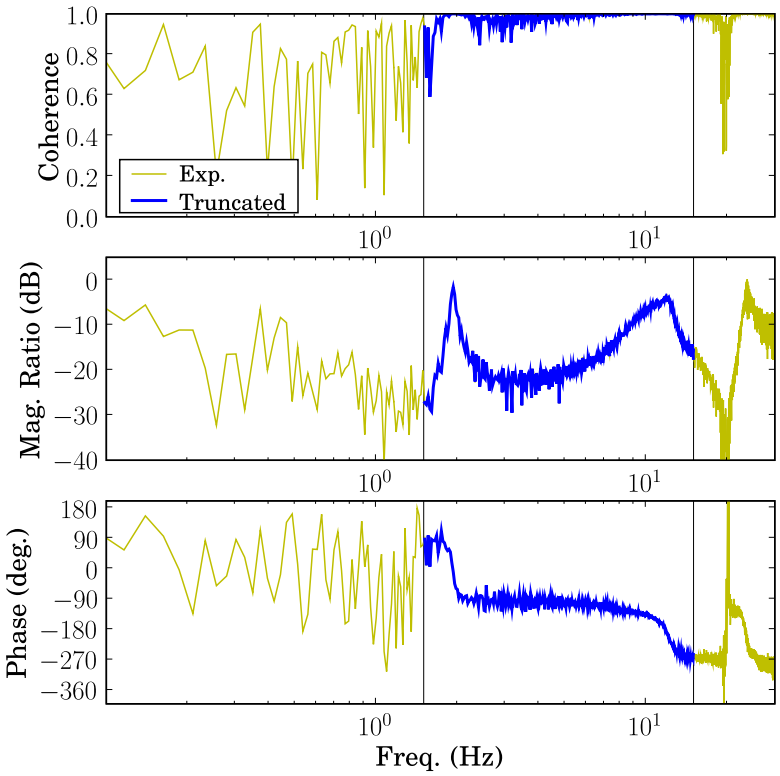

Figure 110: Flexible base Bode plot $\ddot{x}/v$ with coherence showing the truncated range for initial system identification.

7.8.4 General Approach

The general approach to system ID used for SAMII was based on using a Nelder-Mead optimization algorithm to find the values for unknown model parameters that minimize a specified cost. The cost is the squared sum of the error between predicted and experimental Bode responses for θ/v and $\ddot{x}/v$. This error was calculated several different ways including linear magnitude and dB magnitude with and without phase error.

7.8.5 Initial Attempt

An initial attempt at system identification was based on simply taking the truncated data from Figures 109 and 110 and constructing a cost function based on the error between model and experiment:

$$cost = \sum |\text{Exp}(s) - \text{Model}(s)|^2 \tag{225}$$

where Exp(s) and Model(s) are complex valued and $s = 2\pi jf$. The results of this approach are shown in Figures 111 and 112. Clearly this approach leads to poor results for the first mode of Figures 112. It is not immediately obvious how the curve-fit could match mode 1 so poorly. Figures 113 and 114 provide insight into the cause of the poor curve-fitting results. They show Exp(s) and Model(s) from equation (225) for θ/v and $\ddot{x}/v$ plotted with linear magnitude ratios and with linear x-axes. (Figures 111 and 112 have logarithmic x-axes and the use of decibels on the y-axes makes them essentially log-log plots). By fitting all of the truncated data and using the linear magnitude of the Bode plots, equation (225) is essentially fitting the data as depicted in Figures 113 and 114. Figure 114 shows that with a linear x-axis, mode 2 takes up a much broader portion of the curve and as a result it is essentially given a much higher weighting.

Clearly, this initial approach to system identification leads to poor results stemming from a cost function that does not respect the logarithmic nature of Bode plots. Not only is a more intelligent approach needed, but so is a way to quantify the accuracy of the system identification results so that different approaches can be compared.

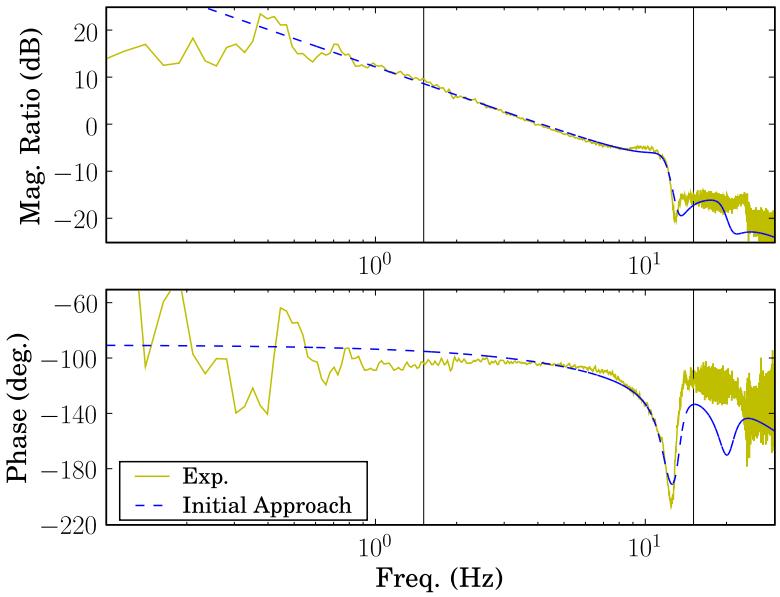

Figure 111: Actuator Bode plot θ/v showing system ID results for an initial attempt based on curve-fitting the complex values for the transfer functions.

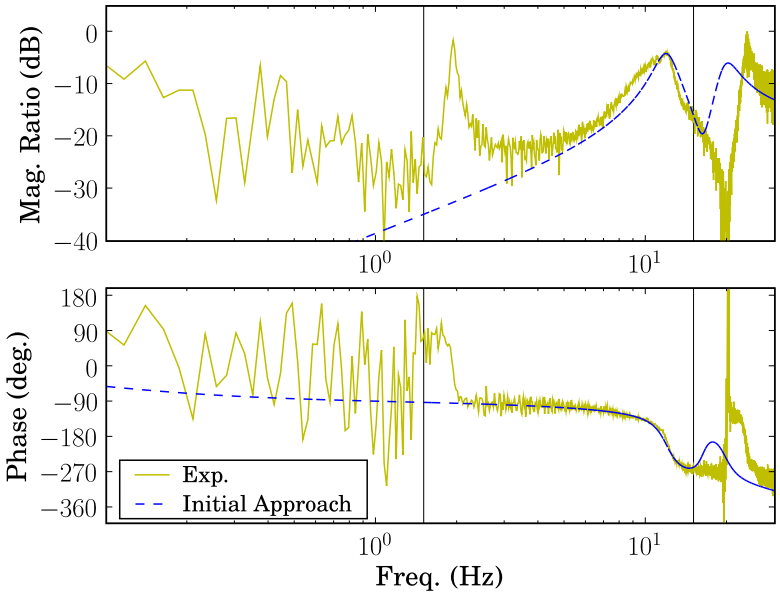

Figure 112: Flexible base Bode plot $\ddot{x}/v$ showing system ID results for an initial attempt based on curve-fitting the complex values for the transfer functions.

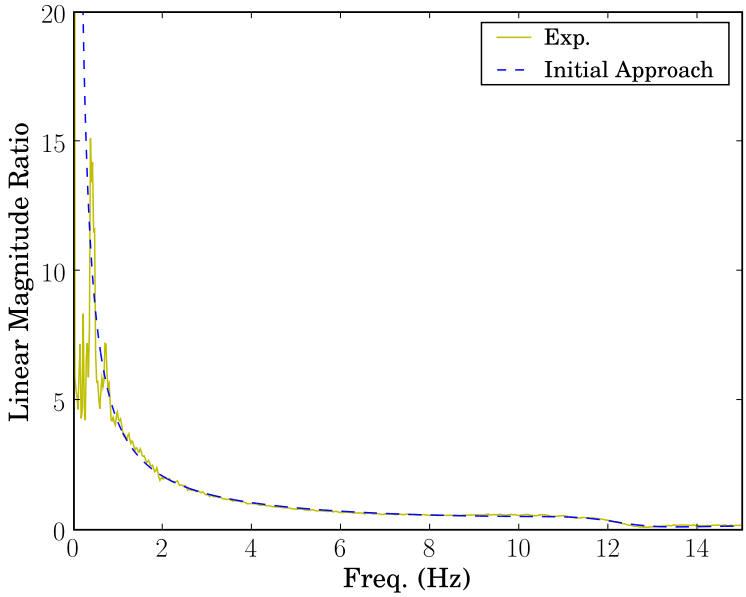

Figure 113: Linear magnitude ratio of the actuator Bode plot θ/v plotted with a linear x-axis.

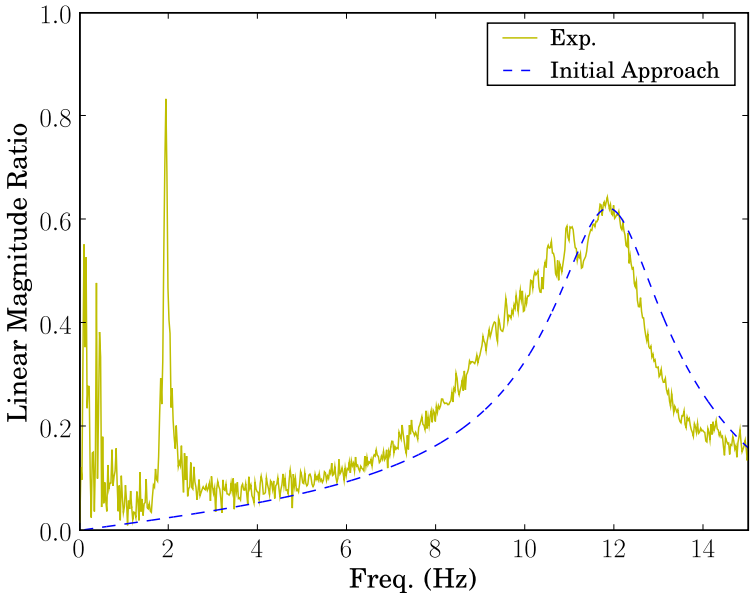

Figure 114: Linear magnitude ratio of the flexible based Bode plot $\ddot{x}/v$ plotted with a linear x-axis.

7.8.6 Quantifying the Accuracy of System ID Results

The goal of this work is to facilitate control design. The accuracy of system ID results will be measured by the degree to which those results can be used to predict open and closed-loop Bode plots, which will be used in control design. This will be measured by the sum of the squared errors between predicted and experimental Bode plots for θ/v, $\ddot{x}/v$, θ/θ_d, $\ddot{x}/\theta_d$, $\theta/\hat{\theta}_d$, and $\ddot{x}/\hat{\theta}_d$. These Bode plots capture the response of the actuator and flexible base for the open-loop system, the system with motion control only (θ feedback as shown in Figure 28), and the system with motion control plus vibration suppression (θ and $\ddot{x}$ feedback as shown in Figure 21).

Note that only the open-loop data will be used during curve-fitting. The the system ID approach is based on finding system the unknown system parameters from curve-fitting only the open-loop data. The accuracy of the parameters will be assessed by using them to predict the closed-loop responses.

In order to account for the logarithmic nature of Bode plots, the error measurement will be based on points with a logarithmic spacing along the frequency axis and will use the difference in dB magnitude along with the difference in phase to calculate the error. The phase error will be multiplied by a weighting factor of 0.1 so that it does not dominate the error. (Typical Bode plots for this system span about 60 dB while the phases can span from –360° to +360°). Figures 115 and 116 show open-loop Bode plots for the actuator and flexible base (θ/v and $\ddot{x}/v$) with the logarithmically spaced points that will be used for the accuracy quantification. Figures 117 and 118 show similar Bode plots for the closed-loop response with θ feedback only (motion control). Figures 119 and 120 show the same Bode plots with θ and $\ddot{x}$ feedback (motion control plus vibration suppression).

Figures 115–120 are used to quantify the accuracy of the initial approach to system ID. Table 5 summarizes the results. It also notes the time to run the optimization. This approach can now be compared to other approaches based on quantitative measures of its accuracy.

Table 5: Quantification of the accuracy of the initial system identification attempt.

Approaches	Open-Loop Fit Error	Closed-Loop Fit Error θ feedback	Closed-Loop Fit Error θ and $\ddot{x}$ feedback	Total Error	Fit Time (sec.)
Initial	19426.00	21100.26	19832.82	60359.08	66.22

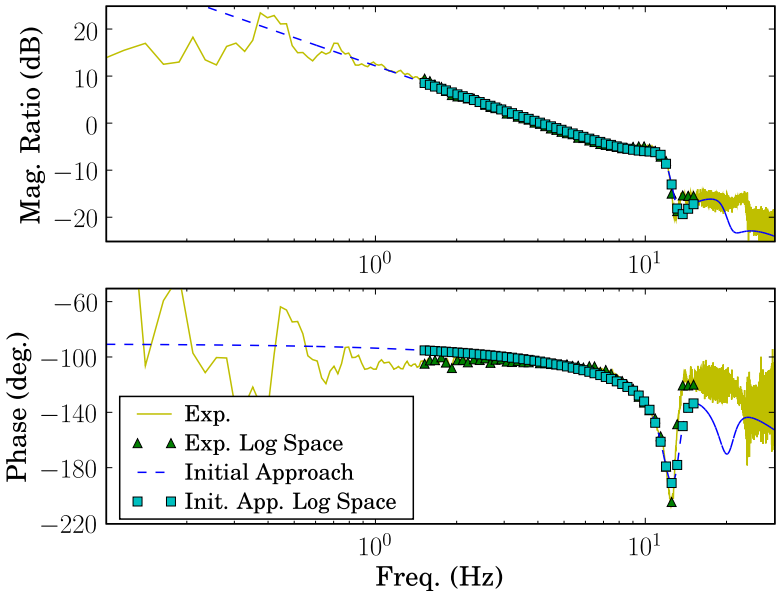

Figure 115: Open-loop actuator Bode plot θ/v showing logarithmically spaced points along the frequency axis that will be used to quantify the accuracy of system identification.

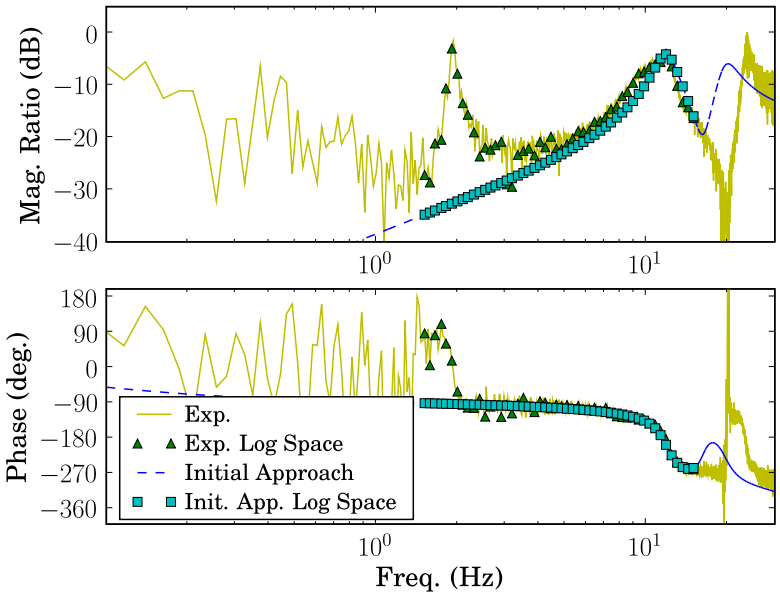

Figure 116: Open-loop flexible base Bode plot $\ddot{x}/v$ showing logarithmically spaced points along the frequency axis that will be used to quantify the accuracy of system identification.

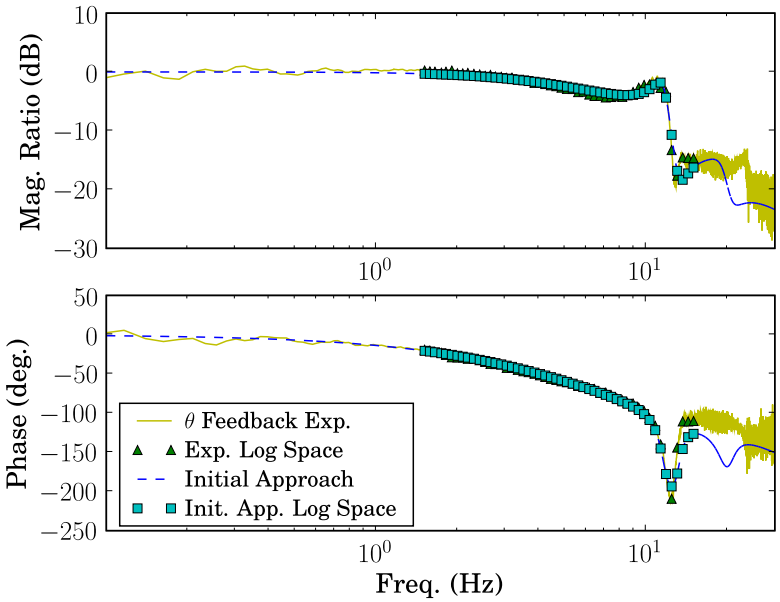

Figure 117: Closed-loop actuator Bode plot θ/θ_d (θ feedback) showing logarithmically spaced points along the frequency axis that will be used to quantify the accuracy of system identification.

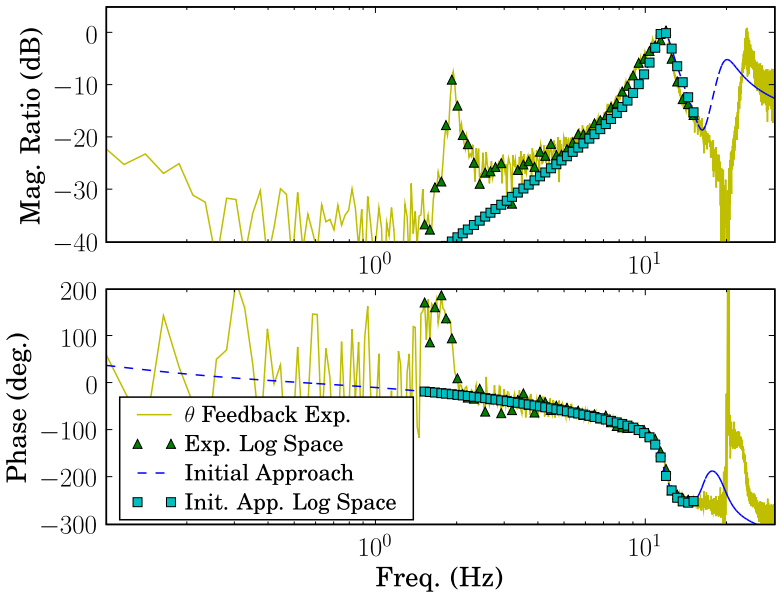

Figure 118: Closed-loop flexible base Bode plot $\ddot{x}/\theta_d$ (θ feedback) showing logarithmically spaced points along the frequency axis that will be used to quantify the accuracy of system identification.

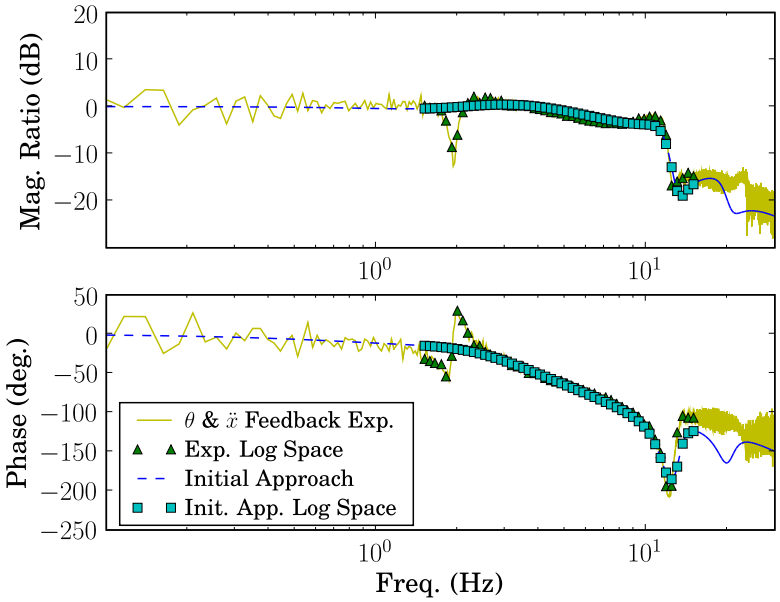

Figure 119: Closed-loop actuator Bode plot $\theta/\hat{\theta}_d$ (θ and $\ddot{x}$ feedback) showing logarithmically spaced points along the frequency axis that will be used to quantify the accuracy of system identification.

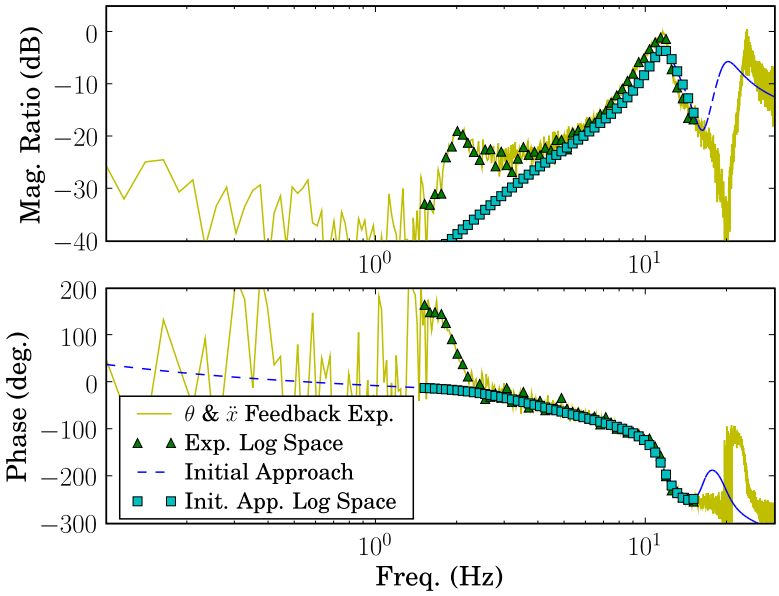

Figure 120: Closed-loop flexible base Bode plot $\ddot{x}/\hat{\theta}_d$ (θ and $\ddot{x}$ feedback) showing logarithmically spaced points along the frequency axis that will be used to quantify the accuracy of system identification.

7.8.7 Additional Approaches and Down-sampling

Additional approaches to curve-fitting were undertaken in light of the fact that the initial attempt essentially ignored the first mode of the system. The next approach was to use dB magnitude and phase explicitly in the cost function. That approach led to much better results, but could still be improved upon by down-sampling the data so that the data used in the cost function is logarithmically spaced along the frequency axis. Figures 121 and 122 show the open-loop Bode plots with the logarithmically spaced data that will be used for this approach to system identification.

Figures 123–128 show that the results with and without the logarithmic down-sampling are very similar. Table 6 compares the quantitative error measurements and run times for these two approaches to one another and the initial approach. Both of the approaches based on dB errors are a significant improvement in accuracy over the initial approach. They also give results fairly close to one another. Note that the logarithmic down-sampling (Log Space dB) cuts the time to run optimization down to 1/3 that of the approach that fits all

Table 6: Quantitative comparison of all approaches to system identification.

Approaches	Open-Loop Fit Error	Closed-Loop Fit Error θ feedback	Closed-Loop Fit Error θ and $\ddot{x}$ feedback	Total Error	Fit Time (sec.)
Initial	19426.00	21100.26	19832.82	60359.08	66.22
dB	1860.55	1889.72	1432.48	5182.76	42.11
Statistical	2177.71	2057.20	1718.50	5953.41	31.41
Log Space dB	1798.24	1761.75	1555.24	5115.23	10.69
Var. Log Space dB	1835.98	1825.28	1476.80	5138.06	9.88

that data (dB). This reduction in computation time could become even more important in cases with significantly more data or more complicated models.

While the results with and without logarithmic down-sampling (dB vs. Log Space dB) are very similar throughout most of the Bode plots, careful inspection of Figure 125 shows that the logarithmic down-sampling sampling curve-fit has lead to slightly worse results around 10 Hz (near the second natural frequency of the structure). The response of the system near resonance is particularly important and that is where the $\ddot{x}$ data is most reliable as shown by the coherence plots.

An additional approach to system ID will be to keep more data near system resonances, effectively weighting these regions more heavily in the cost function. Figures 129 and 130 show Bode plots with variable logarithmic down-sampling of the data. Over the regions from 1.5–2.5 Hz and 9–15 Hz (the regions near the first two structural resonances), the logarithmic down-sampling keeps 100 points per decade. In the region in between the two resonances (2.5–9 Hz), only 20 points per decade are kept.

Results from uniform and variable logarithmic down-sampling are compared in Figures 131–136 and in Table 6. Close inspection of Figure 133 near 10 Hz shows that this approach leads to better results for this plot than uniform logarithmic down-sampling.

7.8.8 Final Approach

The final approach for system ID of SAMII used data that was logarithmically downsampled with two different downsampling rates: 100 points per decade for the ranges 1.5–2.5 Hz and

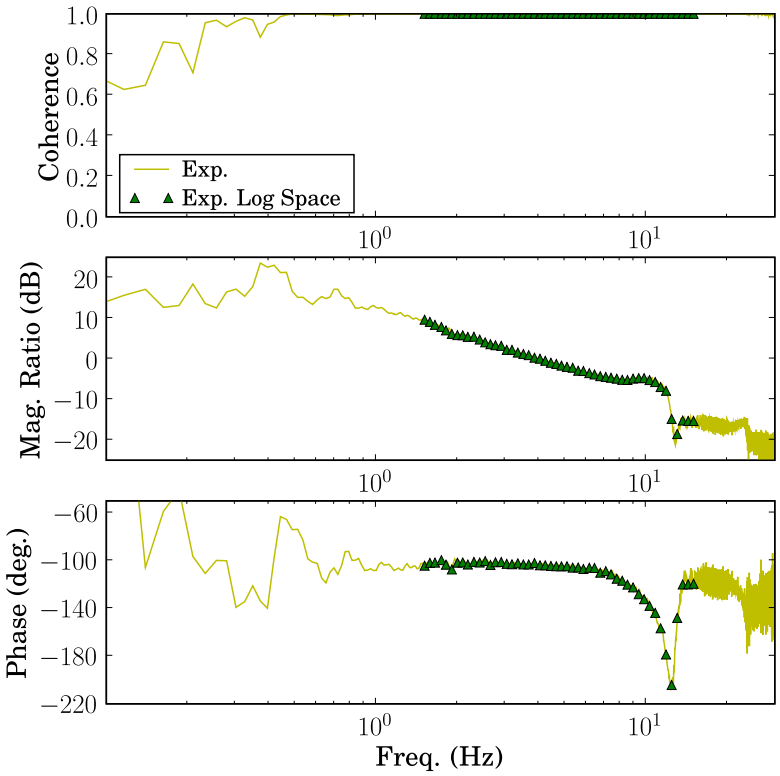

Figure 121: Combined Bode and coherence plot for the actuator θ/v showing logarithmically spaced data points that will be used for system identification.

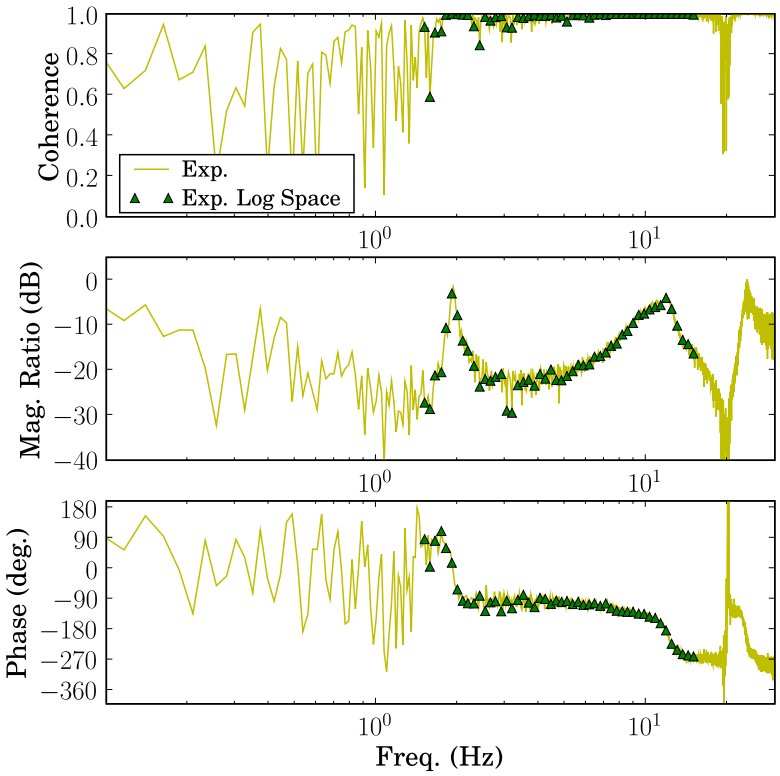

Figure 122: Combined Bode and coherence plot for the flexible base $\ddot{x}/v$ showing logarithmically spaced data points that will be used for system identification.

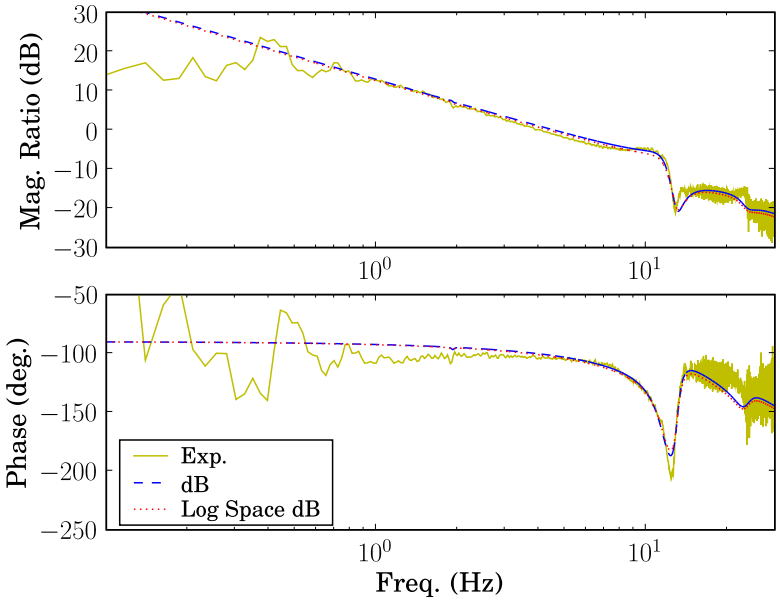

Figure 123: Open-loop actuator Bode plot θ/v comparing curve-fitting results with and without logarithmic down-sampling of the data.

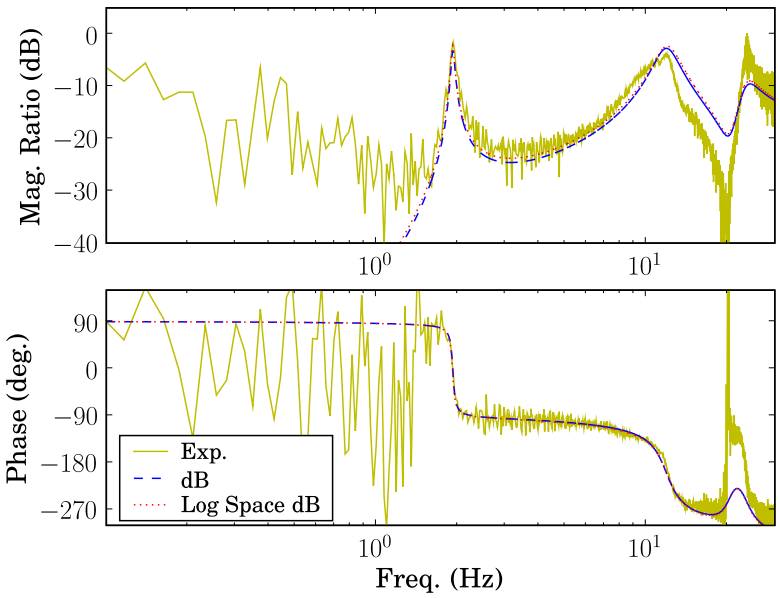

Figure 124: Open-loop flexible base Bode plot $\ddot{x}/v$ comparing curve-fitting results with and without logarithmic down-sampling of the data.

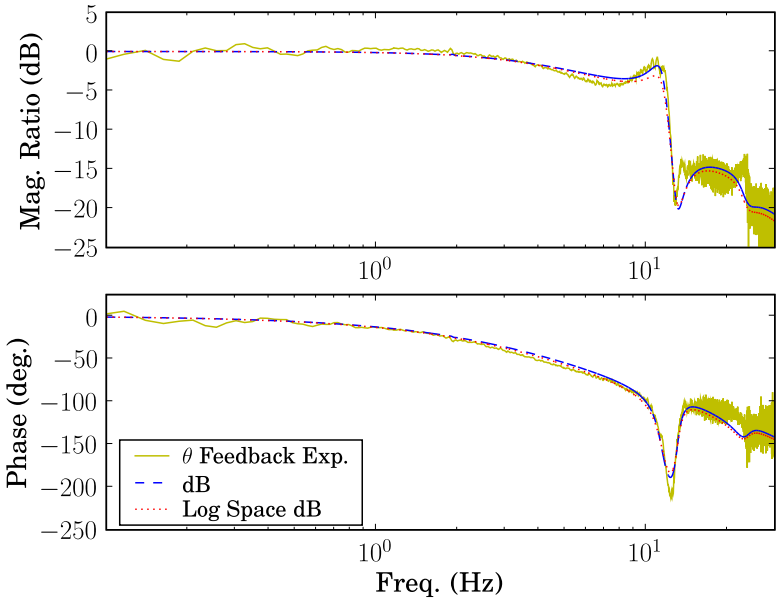

Figure 125: Closed-loop actuator Bode plot θ/θ_d with θ feedback comparing curve-fitting results with and without logarithmic down-sampling of the data.

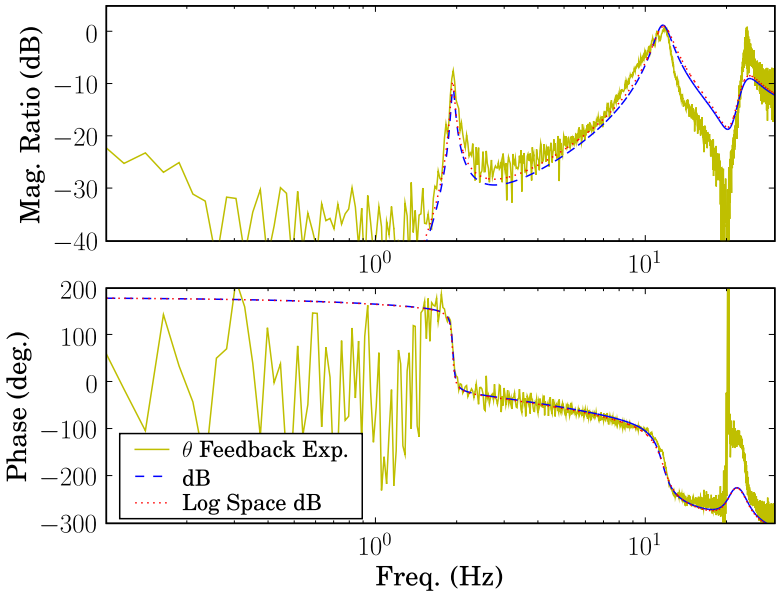

Figure 126: Closed-loop flexible base Bode plot $\ddot{x}/\theta_d$ with θ feedback comparing curve-fitting results with and without logarithmic down-sampling of the data.

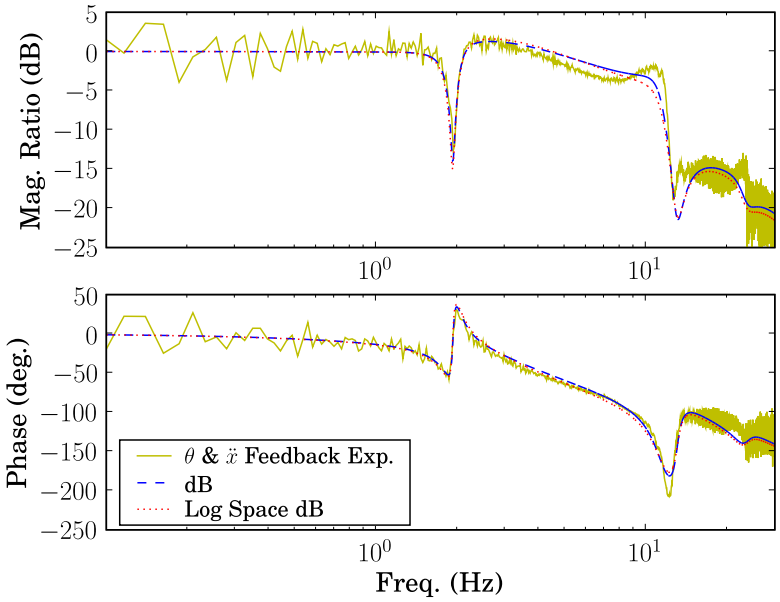

Figure 127: Closed-loop actuator Bode plot $\theta/\hat{\theta}_d$ with θ and $\ddot{x}$ feedback comparing curve-fitting results with and without logarithmic down-sampling of the data.

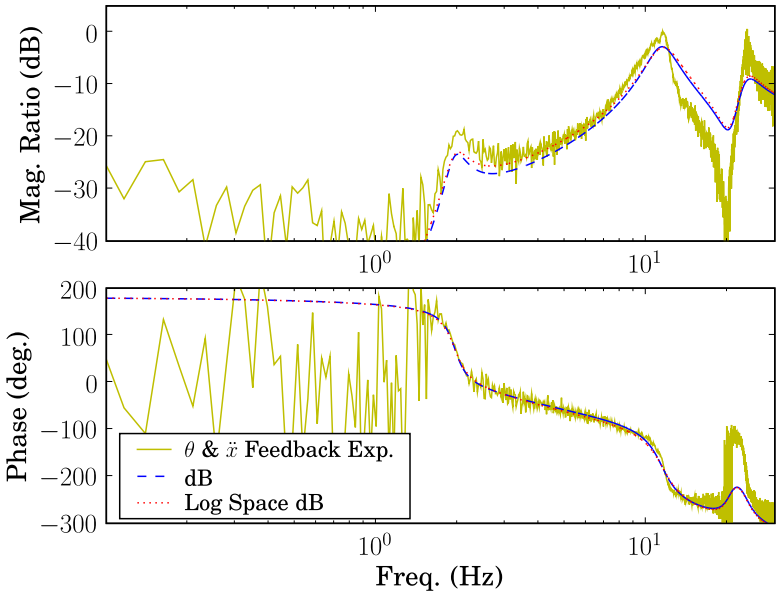

Figure 128: Closed-loop flexible base Bode plot $\ddot{x}/\hat{\theta}_d$ with θ and $\ddot{x}$ feedback comparing curve-fitting results with and without logarithmic down-sampling of the data.

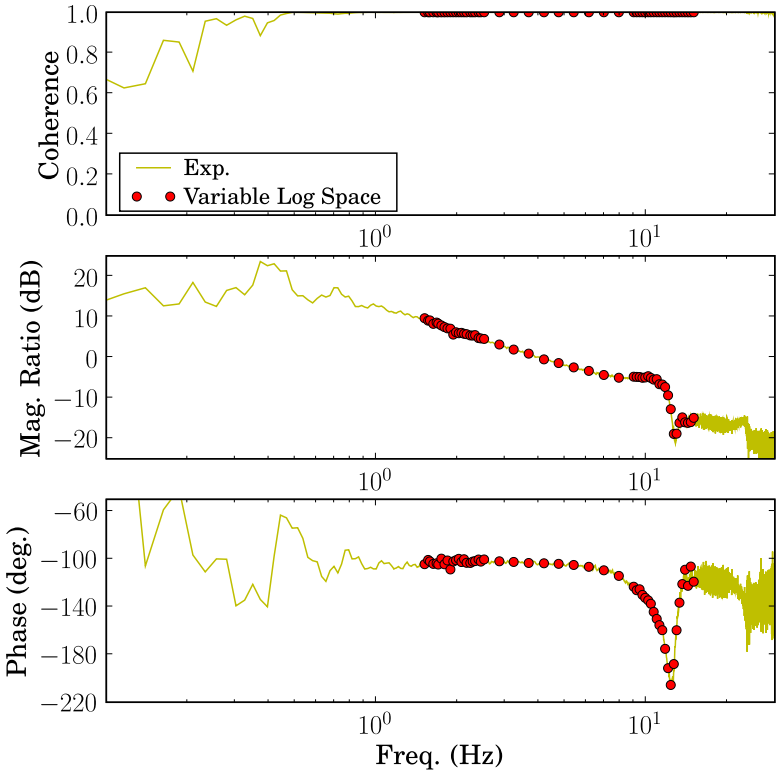

Figure 129: Combined Bode and coherence plot for the actuator θ/v showing logarithmically spaced data points with more points per decade in regions near system resonances.

9–15 Hz and 20 points per decade in the range 2.5–9 Hz. The two Bode outputs (θ/v and $\ddot{x}/v$) were weighted equally. The phase error was multiplied by a weighting factor of 0.1. The magnitude error was calculated based on the difference in dB values between model and experiment. The dB magnitude error had a unity weighting factor.

7.8.9 Conclusions

The approach presented here is based on engineering judgment and the details apply only to system ID of SAMII. Some aspects of this approach may be useful in identification of other systems including using dB magnitudes in error calculations, logarithmic data downsampling, and emphasizing the frequency ranges near system resonances.

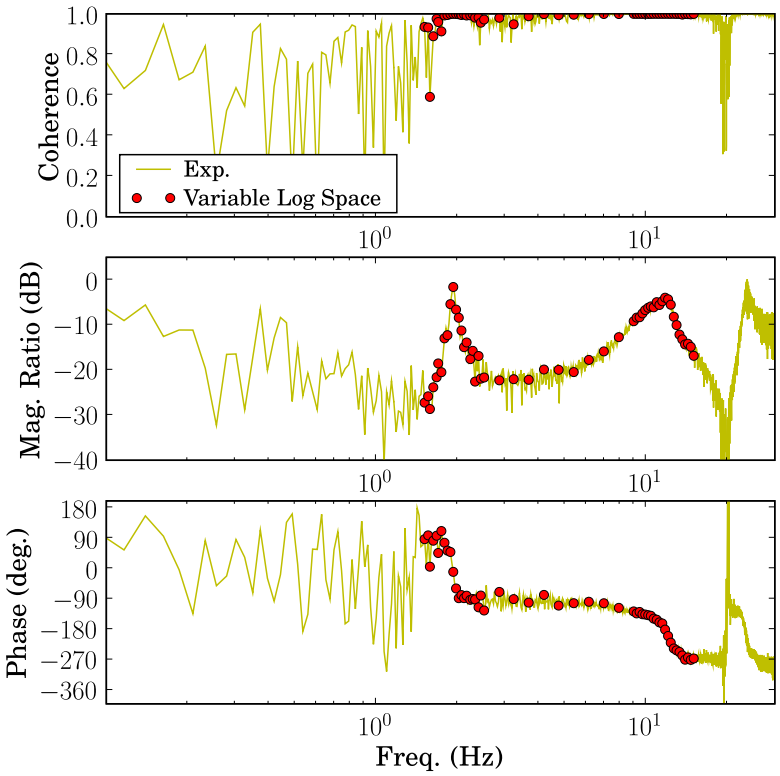

Figure 130: Combined Bode and coherence plot for the flexible base $\ddot{x}/v$ showing logarithmically spaced data points with more points per decade in regions near system resonances.

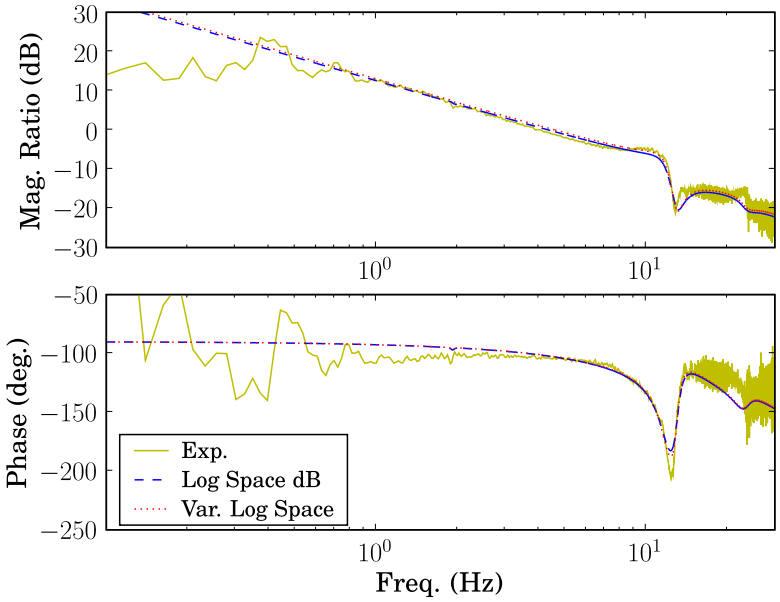

Figure 131: Open-loop actuator Bode plot θ/v comparing curve-fitting results from uniform and variable logarithmic down-sampling of the data.

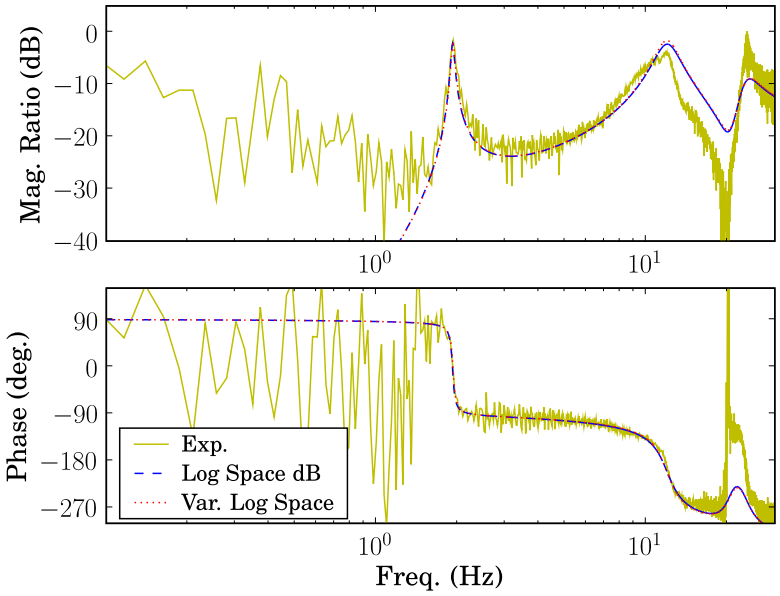

Figure 132: Open-loop flexible base Bode plot $\ddot{x}/v$ comparing curve-fitting results from uniform and variable logarithmic down-sampling of the data.

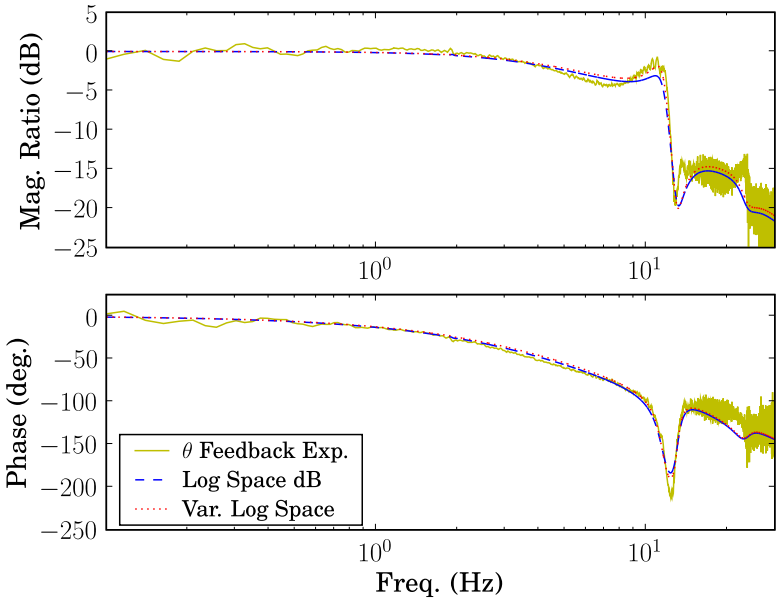

Figure 133: Closed-loop actuator Bode plot θ/θ_d with θ feedback comparing curve-fitting results from uniform and variable logarithmic down-sampling of the data.

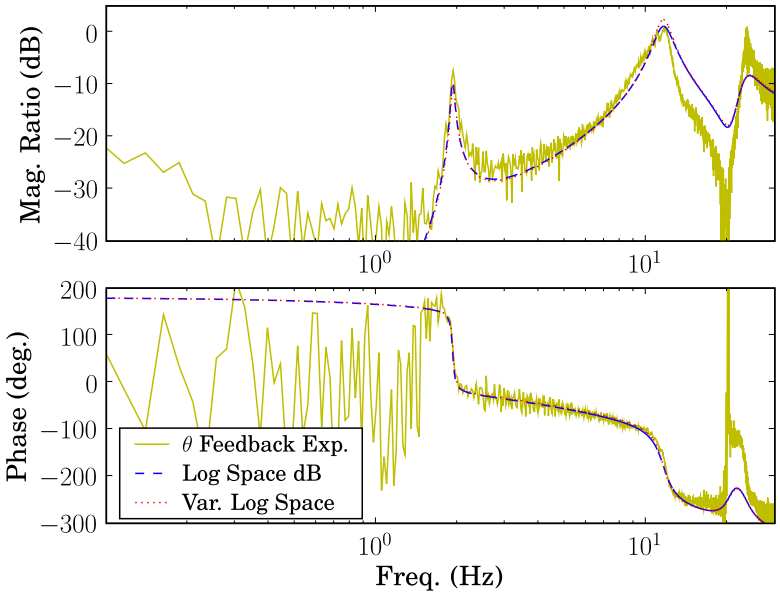

Figure 134: Closed-loop flexible base Bode plot $\ddot{x}/\theta_d$ with θ feedback comparing curve-fitting results from uniform and variable logarithmic down-sampling of the data.

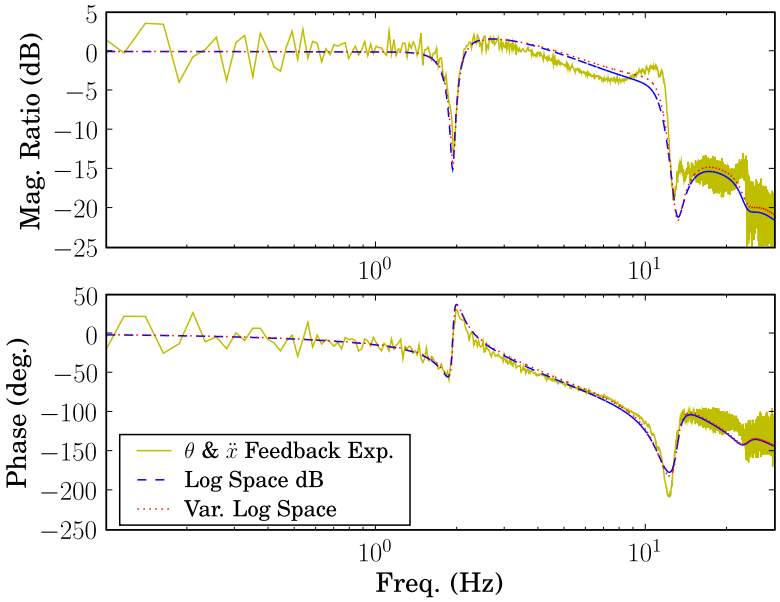

Figure 135: Closed-loop actuator Bode plot $\theta/\hat{\theta}_d$ with θ and $\ddot{x}$ feedback comparing curve-fitting results from uniform and variable logarithmic down-sampling of the data.

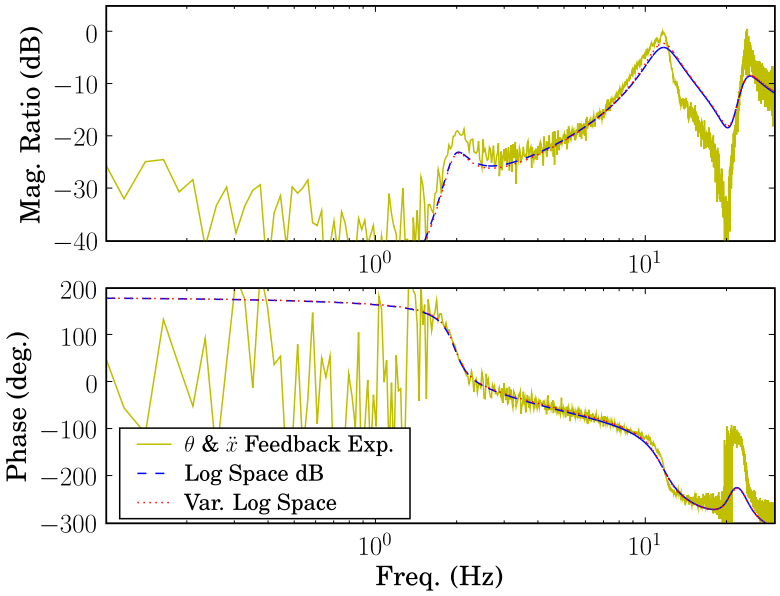

Figure 136: Closed-loop flexible base Bode plot $\ddot{x}/\hat{\theta}_d$ with θ and $\ddot{x}$ feedback comparing curve-fitting results from uniform and variable logarithmic down-sampling of the data.

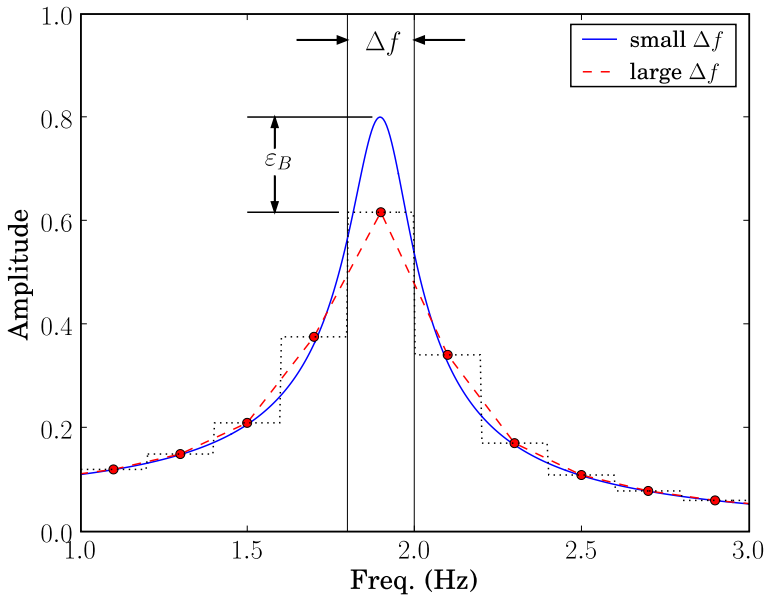

Figure 137: An illustration of bias error for the Bode plot of a lightly damped system.

7.9 Statistical Analysis of Bode Data

7.9.1 Bias Error

Statistical errors in the measurement of frequency domain data come in two forms: bias errors and random errors [6]. Bias errors are systematic in nature and will be repeated from test to test (i.e. they are not random). Figure 137 illustrates the problem. The Fourier transform of data with a finite record length T leads to the averaging of frequency domain values over frequency intervals with $\Delta f = 1/T$. This averaging can lead to significant errors, particularly near the peaks of lightly damped Bode plots.

Bendat and Peirsol [6, p. 267] state the the bias error for a second-order system can be approximated by

$$\varepsilon_B(f_r) \approx \frac{-1}{3}\left(\frac{B_e}{B_r}\right)^2 \tag{226}$$

where f_r is a resonant frequency, $B_e = \Delta f = 1/T$, and B_r is the half-power bandwidth $B_r \approx 2\zeta f_r$.

For system identification of SAMII, bias error will be the worst at the first mode. $f_1 \approx 1.9$ Hz and $\zeta_1 \approx 0.0175$, which leads to $B_r = 0.0665$. If bias error of less than 1% is desired ($\varepsilon_B = 0.01$), $B_e = 0.0115$ and $T = 86.82$ second. In order to accurately measure the peak

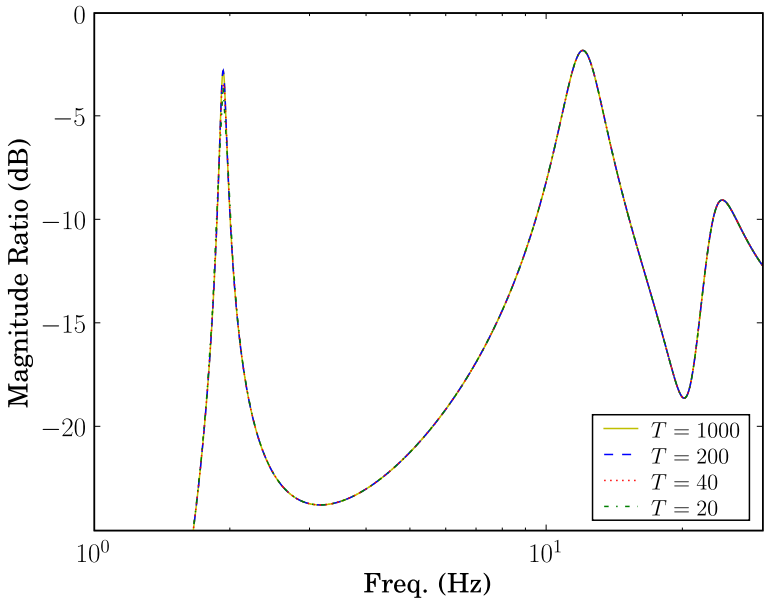

Figure 138: Bode plot of SAMII's flexible base $\ddot{x}/v$ with varying T.

of SAMII's first mode, data should be taken with $T = 100$ seconds. The only problem with this is that $T = 100$ will lead to much smaller Δf than necessary for reasonable bias errors over much of the frequency range. Figures 138–140 illustrate this.

Figure 138 shows the Bode plot generated with $T = 1000$, 200, 40, and 20 seconds. Note that except for the region around the first mode (1.7–2.1 Hz), there is no noticeable difference between these plots. Figure 139 shows the bias error over the 1–30 Hz frequency range for $T = 40$ seconds. Figure 140 zooms in on the y-axis of Figure 139, showing that $\varepsilon_B < 0.01$ everywhere except in the frequency range 1.7–2.1 Hz. The approach will be to collect data with $T = 100$ seconds and then to average the data across frequencies so that $T = 40$ at frequencies greater than 2.1 Hz. This averaging will reduce noise in the Bode plot for these frequencies without causing bias error.

7.9.2 Random Errors

Random errors refer to scatter in the data usually caused by noise of some form. For Bode plots, the random error is closely related to the coherence γ_{xy} between the input x and the

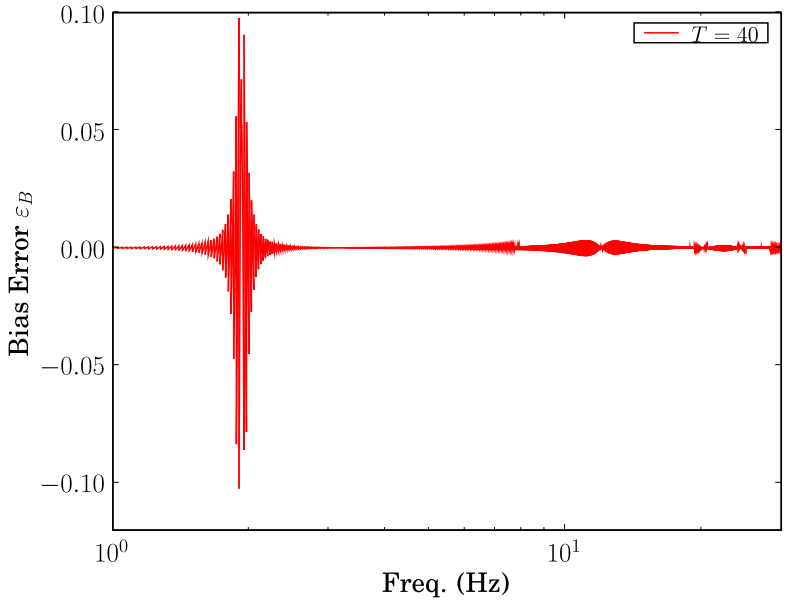

Figure 139: Bias error ε_B for $\ddot{x}/v$ with $T = 40$ seconds.

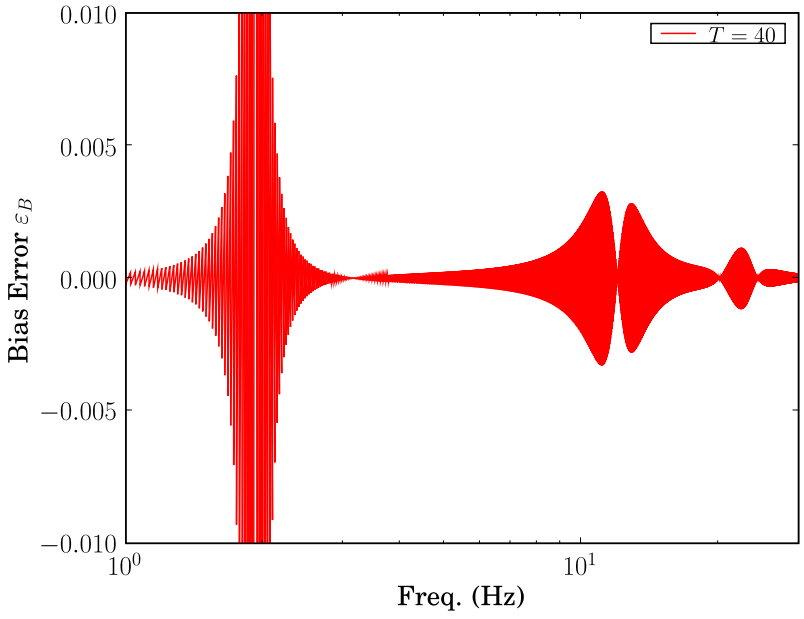

Figure 140: Bias error ε_B for $\ddot{x}/v$ with $T = 40$ seconds (zooming in on Figure 139).

output y of the system:

$$\varepsilon_R = \frac{[1-\gamma_{xy}^2(f)]^{0.5}}{|\gamma_{xy}(f)|\sqrt{2n_d}} \tag{227}$$

where n_d refers to the number of independent sub-records. Figure 141 shows the Bode plot for $\ddot{x}/v$ for this approach. Three tests of 100 seconds each where run. Bode plots from these 3 tests were averaged in the frequency domain to give one test with $n_d = 3$. For frequencies greater than 2.1 Hz, this average test was then averaged across 5 frequency steps (essentially down-sampling the data by a factor of 5, averaging over each interval). This leads to $n_d = 15$ for frequencies above 2.1 Hz. The corresponding random error ε_R is shown in Figure 142. Note that the random relative error is less than 0.1 (10%) for most of the frequency range from 1.5–15 Hz.

The 95% confidence intervals for magnitude and phase are given by

$$\begin{aligned} |\hat{H}|(1-2\varepsilon_R) &< |\hat{H}| < |\hat{H}|(1+2\varepsilon_R) \\ \hat{\phi}(1-2\varepsilon_R) &< \hat{\phi} < \hat{\phi}(1+2\varepsilon_R) \end{aligned} \tag{228}$$

where $|\hat{H}|$ refers to the estimate of the magnitude of the Bode plot and $\hat{\phi}$ refers to the estimate of the phase. A Bode plot that includes the 95% confidence intervals is shown in Figure 143. Note that there is a strong correlation between coherence and ε_R

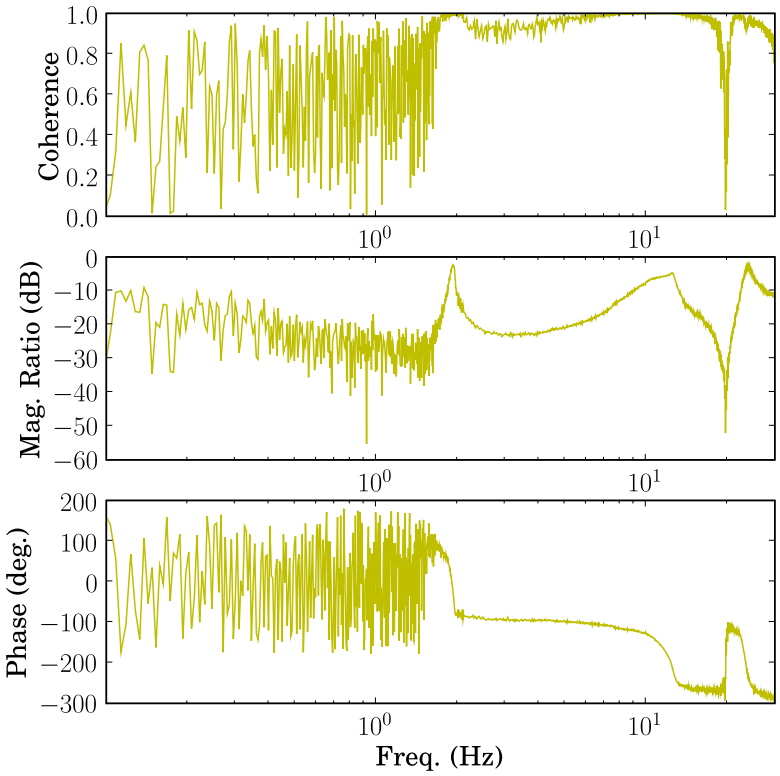

Figure 141: Bode plot with coherence for $\ddot{x}/v$ with 3 tests of $T = 100$ seconds and averaging across 5 frequency steps for $f > 2.1$ Hz.

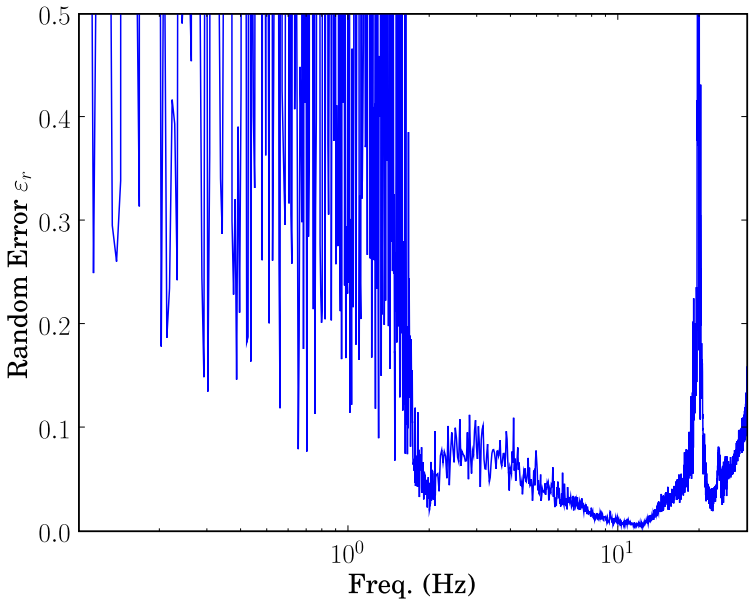

Figure 142: Random error ε_R corresponding to the Bode plot of Figure 141.

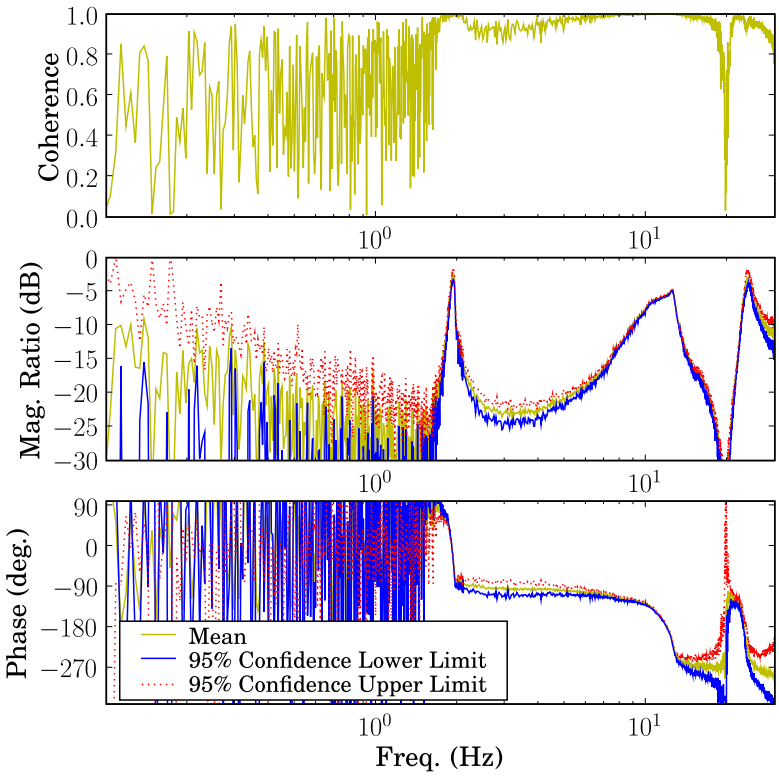

Figure 143: Bode plot with coherence for $\ddot{x}/v$ showing the 95% confidence interval.

CHAPTER VIII

NUMERICAL CONCERNS

8.1 Introduction

There are two primary areas where questions about numeric implementation may significantly affect the usefulness of the TMM. One is floating-point accuracy and the other is repeated roots of the characteristic determinant. The floating-point accuracy problem deals with whether or not the natural frequencies found by the TMM are valid. Handling repeated or nearly repeated roots correctly will be important when trying to find all of the natural frequencies and mode shapes of systems with symmetry or some other reason why two or more natural frequencies would be very nearly identical.

8.2 Floating-Point Accuracy

The limitations of floating-point accuracy can be a significant obstacle to practical implementation of the TMM, especially when trying to work with higher modes. One calculation where this is potentially problematic is trying to find small differences between large numbers. This occurs while trying to find the natural frequencies of systems that include beam elements. Natural frequencies of a system modeled with the TMM are found by numerically searching for roots of the characteristic determinant. These determinants can involve terms like $\sinh\beta - \sin\beta$. As β becomes large, $\sin\beta$ is negligible compared to $\sinh\beta$.

8.2.1 An Example

As an example, consider a system made up of a pinned-pinned beam (a pinned-pinned beam is analyzed by Pestel and Leckie as a system with potential numeric problems [49,

pp. 192-213]). The transfer matrix for a beam element is given by

$$\mathbf{U}_{\text{beam}} = \begin{bmatrix} c_1 & \frac{c_4\, L}{2\,\beta} & -\frac{a\, c_3}{2\,\beta^2} & -\frac{a\, c_2\, L}{2\,\beta^3} \\ \frac{\beta\, c_2}{2\, L} & c_1 & \frac{a\, c_4}{2\,\beta\, L} & \frac{a\, c_3}{2\,\beta^2} \\ -\frac{\beta^2\, c_3}{2\, a} & \frac{\beta\, c_2\, L}{2\, a} & c_1 & -\frac{c_4\, L}{2\,\beta} \\ -\frac{\beta^3\, c_4}{2\, a\, L} & \frac{\beta^2\, c_3}{2\, a} & -\frac{\beta\, c_2}{2\, L} & c_1 \end{bmatrix} \tag{229}$$

where

$$a = \frac{L^2}{EI} \tag{230}$$

$$\beta = \left(-\frac{L^4\, \mu\, s^2}{EI}\right)^{1/4} \tag{231}$$

$$c_1 = 0.5\, \cosh\beta + 0.5\, \cos\beta \tag{232}$$

$$c_2 = \sinh\beta - \sin\beta \tag{233}$$

$$c_3 = \cos\beta - \cosh\beta \tag{234}$$

$$c_4 = \sinh\beta + \sin\beta \tag{235}$$

The state vector is

$$\mathbf{z} = \begin{Bmatrix} x \\ \theta \\ M \\ V \end{Bmatrix} \tag{236}$$

For a pinned-pinned beam, the TMM model is

$$\begin{Bmatrix} 0 \\ \theta \\ 0 \\ V \end{Bmatrix}_0 = \mathbf{U}_{\text{beam}} \begin{Bmatrix} 0 \\ \theta \\ 0 \\ V \end{Bmatrix}_1 \tag{237}$$

The first and third row of equation (237) give

$$\begin{Bmatrix} 0 \\ 0 \end{Bmatrix} = \begin{bmatrix} \mathbf{U}_{\text{beam12}} & \mathbf{U}_{\text{beam14}} \\ \mathbf{U}_{\text{beam32}} & \mathbf{U}_{\text{beam34}} \end{bmatrix} \begin{Bmatrix} \theta_1 \\ V_1 \end{Bmatrix} \tag{238}$$

or

$$\begin{Bmatrix} 0 \\ 0 \end{Bmatrix} = \mathbf{subU} \begin{Bmatrix} \theta_1 \\ V_1 \end{Bmatrix} \tag{239}$$

where

$$\mathbf{subU} = \begin{bmatrix} \mathbf{U}_{\text{beam12}} & \mathbf{U}_{\text{beam14}} \\ \mathbf{U}_{\text{beam32}} & \mathbf{U}_{\text{beam34}} \end{bmatrix} \tag{240}$$

Non-trivial solutions to equation (239) occur only when $|\mathbf{subU}| = 0$, which occurs at system eigenvalues. $|\mathbf{subU}| = 0$ is the characteristic equation of the system and $|\mathbf{subU}|$ is referred to as the characteristic determinant.

For a pinned-pinned beam,

$$\mathbf{subU} = \begin{bmatrix} \dfrac{(\sinh\beta + \sin\beta)\ L}{2\,\beta} & -\dfrac{(\sinh\beta - \sin\beta)\ L^3}{2\,\beta^3\ EI} \\ \dfrac{\beta\ (\sinh\beta - \sin\beta)\ EI}{2\ L} & -\dfrac{(\sinh\beta + \sin\beta)\ L}{2\,\beta} \end{bmatrix} \tag{241}$$

and the characteristic determinant is

$$c = \frac{(\sinh\beta - \sin\beta)^2\ L^2}{4\,\beta^2} - \frac{(\sinh\beta + \sin\beta)^2\ L^2}{4\,\beta^2} \tag{242}$$

Round-off error can be significant when calculating c from the above equation. Algebraic simplification sheds considerable insight on the problem:

$$c = -\frac{\sin\beta\ \sinh\beta\ L^2}{\beta^2} \tag{243}$$

The roots of c really occur when $\sin\beta = 0$, because the only root of $\sinh\beta$ is $\beta = 0$ which is a trivial solution. The problem is that $\sin\beta$ is the term that will be neglected due to round-off error during the calculation of c in equation (242). Note that in a purely numeric implementation of the TMM, the method does not use the symbolic representations of equation (242) or (243). It has only the numeric values of equation (241) and calculates the determinant directly. This introduces even more opportunities for round-off than equation (242).

A symbolic implementation of the TMM may be able to get equation (243) directly and avoid the floating-point problems associated with the subtraction in equation (242) or the direct calculation of the determinant of **subU**.

8.2.2 Floating-Point Subtraction

Knowing when the calculation of $\sinh\beta \pm \sin\beta$ will be a problem depends on understanding a little bit of how floating-point addition and subtraction are preformed and how numbers are represented on the hardware and software being used. For 32-bit machines, floating-point arithmetic is most often performed according to IEEE 754 where there are 23 bits used to encode the mantissa for single precision calculations and 52 bits used for double precision. Even though it is possible to represent very small numbers, the smallest relative difference between numbers is roughly 2^{-23} or 1.1920928955078125e–07 times the magnitude of the larger number for single precision and 2^{-52} or 2.2204460492503131e–16 for double precision.

A simple numerical experiment illustrates the problem. Consider

```
1  m=1.0
2  e=1.0
3  while m+e>m:
4      e=e/2.0
```

Executing this Python code (or a similar syntax in Matlab©) on a 32-bit machine using double precision gives $e = 1.1102230246251565e - -16$, which is exactly $2^{-52}/2$ meaning that 2^{-52} is the smallest number whose difference from 1 can be detected.

Basically, when adding or subtracting two numbers represented with a finite length mantissa, if the mantissas of the two numbers do not overlap, the smaller number has no effect on the result. As a simple example, consider a fictitious computer that uses base 10 for calculations with 3 bits for the mantissa and 2 bits for the exponent. On this computer, the number 10000 would be represented as $1.00e+04$ and the number 1 would be encoded as $1.00e+00$. Subtracting 1 from 10000 exactly would give 9999, but representing this number on the fictitious computer would give $1.00e+4$ because of rounding error. This is the exact same problem a real 32-bit computer faces, but with base 2 and 23 or 52 bits for the mantissa.

8.2.3 Floating-Point Analysis of the Pinned-Pinned Beam

Returning to the analysis of the pinned-pinned beam, a key floating-point question is for what values of β will the effects of $\sin\beta$ in $\sinh\beta \pm \sin\beta$ be ignored? A similar numerical

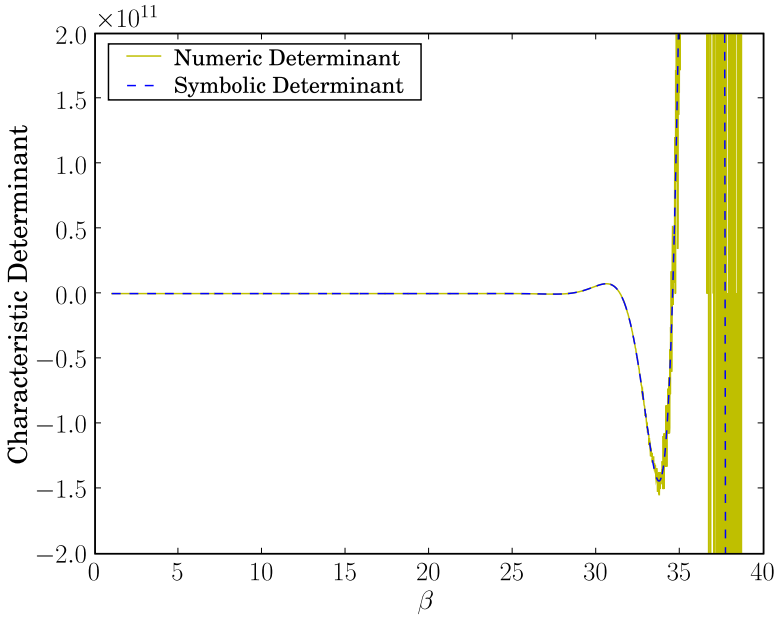

Figure 144: Value of the characteristic determinant from numeric evaluation based on the matrix in equation (241) and from the symbolic expression in equation (243).

experiment to the one preformed previously answers this question:

```
m=1.0
e=1.0
while m+e>m:
    m=m*2.0
```

The above Python code produces $m = 9.00719925e+15$ (again for double precision analysis on a 32-bit machine). When $\sinh\beta > 9.00719925e+15$, the effect of $\sin\beta$ will be lost. This corresponds to $\beta > 37.429948$. For a pinned-pinned beam, the natural frequencies are $r\pi$, so that this β would be exceeded when seeking the root of the 12^{th} mode ($37.429948/\pi = 11.91$).

Figure 144 compares the characteristic determinant from numeric evaluation of the determinant of the matrix of equation (241) with that from the symbolic expression of equation (243). The numeric determinant starts to show problems with round-off noise before the value of β gets to 37.429948. This is not surprising since there are more opportunities for round-off error in the intermediate calculations of the determinant besides the evaluation of $\sinh\beta \pm \sin\beta$.

Table 7: Comparison of root finding results: symbolic ($\sinh\beta\sin\beta$) vs. numeric determinant ($|\mathbf{subU}|$).

Mode Number	Theoretical β	Symbolic Analysis	Numeric Determinant	Symbolic Error (%)	Numeric Error (%)
1	3.1416	3.1416	3.1416	1.414e–14	8.447e–5
2	6.2832	6.2832	6.2832	1.414e–14	-7.469e–5
3	9.4248	9.4248	9.4248	-1.885e–14	-2.164e–5
4	12.5664	12.5664	12.5664	-1.414e–14	-0.0002338
5	15.7080	15.7080	15.7080	1.131e–14	-0.0002338
6	18.8496	18.8496	18.8496	0	-0.0002338
7	21.9911	21.9911	21.9911	-3.231e–14	0.0002209
8	25.1327	25.1327	25.1327	0	0.000164
9	28.2743	28.2743	28.2745	0	-0.0005875
10	31.4159	31.4159	31.4160	-3.393e–14	-0.0002338
11	34.5575	34.5575	34.5485	0	0.0261
12	37.6991	37.6991	3.1416	-1.885e–14	91.67
13	40.8407	40.8407	3.1416	0	92.31

Table 7 compares the results of using the symbolic equation (243) and the determinant of the numeric equation (241) to find the first 13 natural frequencies of the pinned-pinned beam. The initial guess used in the numerical search algorithm is 1.05 times the known theoretical value. This is a bit contrived because for a more complicated system the natural frequencies will be completely unknown. However, it is convenient for this analysis and serves to illustrate the effects of floating-point arithmetic on the analysis.

The primary point is that the numeric analysis fails to converge once $\beta > 37.429948$ and that the algebraic rearranging of the symbolic analysis solves this problem.

8.2.4 A Pragmatic Approach

It is very difficult to make general statements about when floating-point arithmetic will significantly affect TMM analysis. To do so would require knowing all of the possible forms that a characteristic determinant might take for any system the TMM might be used to model. That seems nearly impossible. But there are several fairly simple steps that can be taken to check for these kinds of problems.

Round-off error may lead to two separate problems that the user must be aware of. The first is that if equation (242) is used, all values for $\beta > 37.429948$ will be roots. The second

is that if the determinant of **subU** from equation (241) is used, there will be round-off error of the order of $2^{-52} \sinh \beta$. This round-off error will show up as numeric noise on a plot of the characteristic determinant. Plotting the characteristic determinant as a function of β should reveal either of these problems. Another way to test the validity of a root returned by a numeric search algorithm would be to verify that with other nearby points as initial guesses the search algorithm converges to the same root. Finally, if the value of the characteristic determinant really is very small at the frequency determined, the root probably is valid.

8.2.5 Symbolic Analysis

Using the symbolic capabilities that are integrated into the analysis package provides significant protection against floating-point problems. This is illustrated by the symbolic columns of Table 7. Note that the roots found using symbolic analysis are accurate to within machine accuracy. The absolute errors are of the order 2e–16 times the magnitude of the theoretical answer (2e–16 $\approx 2^{-52}$).

There is no reason not to use this symbolic capability. It takes no additional effort on the part of the user, it is faster when searching for natural frequencies, and it is more accurate. It also makes possible inspection of the symbolic form of the characteristic determinant.

Inspecting the symbolic form of the characteristic determinant maybe the only truly reliable way to determine if floating-points errors are going to be significant and if they can be eliminated. Inspection of equation (243) reveals that since $\sinh \beta \neq 0$ for $\beta > 0$, the natural frequencies could actually be found from the roots of $\sin \beta = 0$, which is very well conditioned numerically and actually leads to the closed-form theoretical solution of $\beta = r\pi$.

8.3 Repeated Roots

The second numeric implementation question is how to correctly identify and handle cases where the system has repeated or nearly repeated eigenvalues (roots of the characteristic equation). One example of such a system would be a symmetric or nearly symmetric beam.

8.3.1 Overview

A brief explanation of how the TMM determines system eigenvalues and mode shapes may be helpful. A transfer matrix system model takes the form

$$\mathbf{z}_{\text{end}} = \mathbf{U}_{\text{sys}}(s)\ \mathbf{z}_{\text{base}} \tag{244}$$

(Models in the previous section may have appeared to have $\mathbf{U}_{\text{sys}}(\beta)$, but β is a function of s.)

The system boundary conditions will require that half of the elements of $\mathbf{z}_{\text{end}}$ and $\mathbf{z}_{\text{base}}$ be zero. Partitioning equation (244) appropriately gives

$$\left\{ \begin{matrix} 0 \\ \vdots \\ 0 \end{matrix} \right\} = \mathbf{subU}(s)\hat{\mathbf{z}}_{\text{base}} \tag{245}$$

so that the left-hand side is a column of zeros and $\hat{\mathbf{z}}_{\text{base}}$ refers to the non-zero elements of $\mathbf{z}_{\text{base}}$. For the specific case of a pinned-pinned beam, this would give equation (238). A system eigenvalue is a value of s for which equation (245) has a non-trivial solution and the mode shape comes from the corresponding values of $\hat{\mathbf{z}}_{\text{base}}$. In order for equation (245) to have a non-trivial solution, $\mathbf{subU}(s)$ must have a null space, i.e. it must be rank deficient. If $\mathbf{subU}(s)$ is only rank deficient by 1, then the null space is one dimensional and there is only one system eigenvalue and mode shape corresponding to that value of s.

However, if the rank deficiency is greater than 1, than the null space has dimension equal to the rank deficiency and there are repeated system eigenvalues and more than one mode shape corresponding to that value of s. This is similar to a generalized eigenvalue problem but with the additional requirement that the eigenvalues of $\mathbf{subU}(s) = 0$. It is only the values of s which cause the eigenvalues of $\mathbf{subU}(s)$ to equal 0 that are system eigenvalues, because only if an eigenvalue of $\mathbf{subU}(s)$ equals 0 does equation (245) have a non-trivial solution. Recall that the definition of an eigenvalue is a value for λ such that

$$\mathbf{A}\bar{v} = \lambda\bar{v} \tag{246}$$

Table 8: Parameter values for SAMII's cantilever beam.

Parameter	Value	Units
μ	5.72815	kg/m
L	4.6482	m
EI_1	339137	N×m^2

where $\bar{v}$ is the corresponding eigenvector. For $\lambda = 0$, equation (246) gives

$$\mathbf{A}\bar{v} = 0 \tag{247}$$

which is precisely the form of equation (245).

Consequently, a repeated system eigenvalue is one for which $\mathbf{subU}(s)$ has a null space with dimension greater than one and the system mode shapes corresponding to that value of s span the null space (i.e. form a basis for the null space). In order to get mode shapes that make sense, care must be taken to make sure that the basis vectors are orthogonal. The null space and corresponding basis vectors for $\mathbf{subU}(s)$ could be found using (at least) three different approaches which have different strengths and weaknesses:

1. Row reduction to upper triangular form
2. Eigenvalue/eigenvector analysis
3. Singular value decomposition

8.3.2 Illustrative Examples

To illustrate these ideas, consider the case of a cantilever beam. This analysis will use the beam parameters from SAMII's cantilever as shown in Table 8. The first two system eigenvalues and mode shapes will be studied for a two-dimensional beam model (i.e. one capable of bending about two axes) with values for $EI_2/EI_1 = 1.5$, 1.1, 1.001, and 1.0.

For all three approaches (RREF, eigenvalues/eigenvectors, and SVD), the analysis follows the same procedure:

1. Find the system eigenvalues (s_1 and s_2 for this case)
2. Find the system transfer matrix corresponding to each eigenvalue ($\mathbf{U}_{\text{sys}}(s_1)$ and $\mathbf{U}_{\text{sys}}(s_2)$)

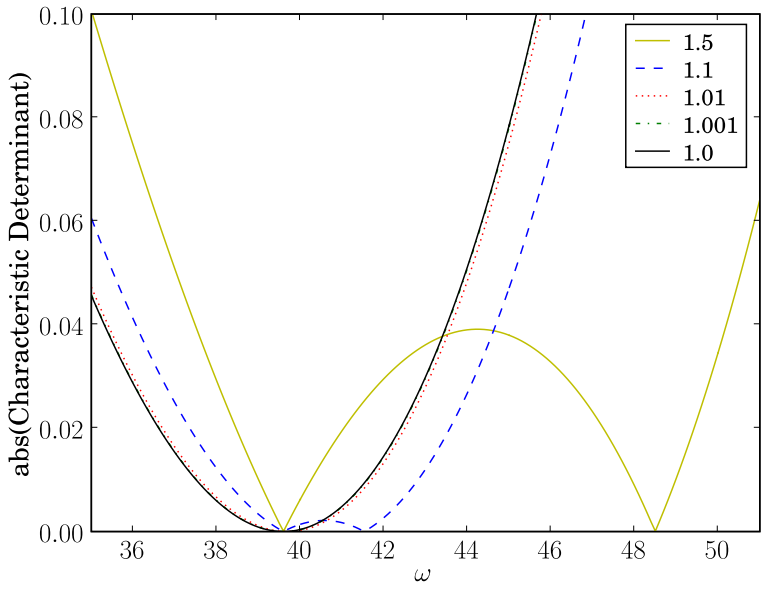

Figure 145: Plot of the characteristic determinant near the first two system eigenvalues for the cases $EI_2/EI_1 = 1.5$, 1.1, 1.01, 1.001, and 1.0

3. Find the system sub-matrix whose determinant is the characteristic equation ($\mathbf{subU}(s_1)$ and $\mathbf{subU}(s_2)$). $\mathbf{subU}(s)$ will come from $\mathbf{U}_{\text{sys}}(s)$ based on system boundary conditions.

4. Find the null space of $\mathbf{subU}(s)$ – this is the only step that differs for the 3 approaches

5. Combined the null space basis vectors with known 0 boundary conditions to get $\mathbf{z}_{\text{base}}$, the state vector at the base of the system

6. Use $\mathbf{z}_{\text{base}}$ and the transfer matrices of individual elements to determine the state vectors at other spatial locations to visualize the mode shape

8.3.2.1 Initial Steps

The analysis begins with finding the system eigenvalues. These values must be searched for numerically, and this requires good initial guesses. These can be found by plotting the characteristic determinant as a function of input frequency ($s = j\,\omega$). Figures 145–147 show the characteristic determinant near the first two natural frequencies at various zoom levels to show the initial guesses that should be used for each value of EI_2/EI_1. Note that

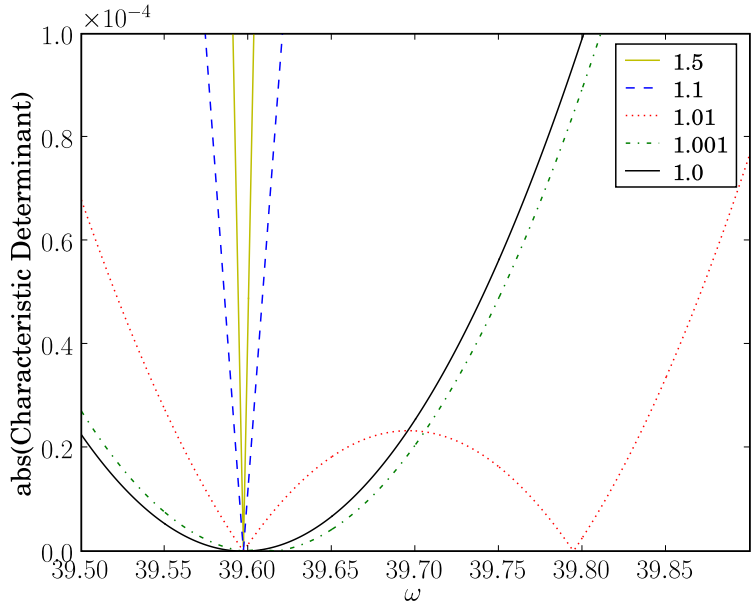

Figure 146: Zooming in on Figure 145 to better show the case $EI_2/EI_1 = 1.01$

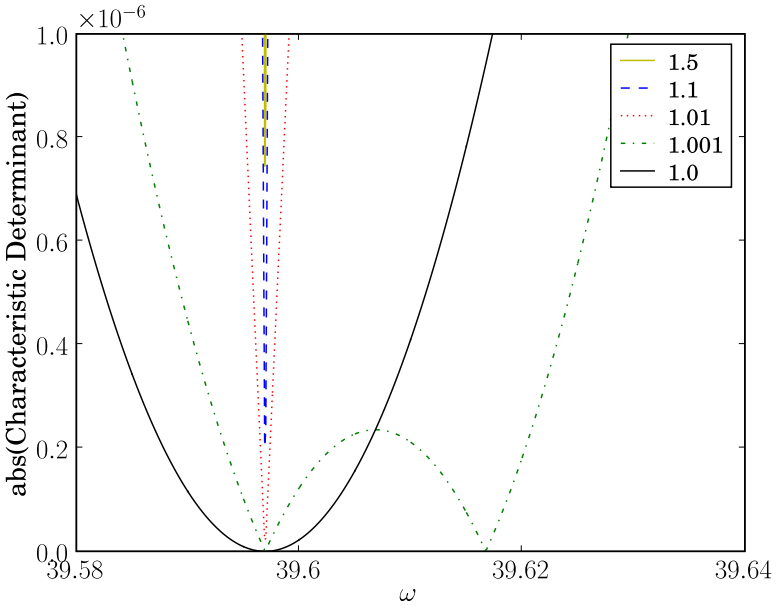

Figure 147: Zooming in further on Figure 146 to better show the cases $EI_2/EI_1 = 1.001$ and 1.0

Table 9: Table of the first two system eigenvalues for each case of EI_2/EI_1 considered (found using a numerical search algorithm).

EI_2/EI_1	s_1	s_2
1.5	$39.5969j$	$48.4962j$
1.1	$39.5969j$	$41.5296j$
1.01	$39.5969j$	$39.7944j$
1.001	$39.5969j$	$39.6167j$
1.0	$39.5969j$	$39.5969j$

a first indicator of a repeated root is that the graph of the characteristic determinant vs. input frequency is tangent to the x-axis at the system eigenvalue.

Table 9 shows the first two system eigenvalues for each value of EI_2/EI_1. Once the system eigenvalues are known, the system transfer matrix $\mathbf{U}_{\text{sys}}(s)$ corresponding to each one can be found. For $s = s_1$ and $EI_2/EI_1 = 1.5$,

$$\mathbf{U}_{\text{sys}}(s_1) = \left[\begin{array}{cccc} 1.345 & 0 & 0 & -3.323e{-}5 \\ 0 & 1.519 & 1.512e{-}5 & 0 \\ 0 & 1.525e{+}5 & 1.519 & 0 \\ -4.462e{+}4 & 0 & 0 & 1.345 \\ 0 & -5.129 & -3.295e{-}5 & 0 \\ 0.2984 & 0 & 0 & -2.172e{-}5 \\ 9.925e{+}4 & 0 & 0 & -4.968 \\ 0 & 1.004e{+}5 & 0.4498 & 0 \end{array}\right. \cdots$$

$$\left.\begin{array}{cccc} 0 & 4.968 & 2.172e{-}5 & 0 \\ -0.4498 & 0 & 0 & 3.295e{-}5 \\ -1.004e{+}5 & 0 & 0 & 5.129 \\ 0 & -9.925e{+}4 & -0.2984 & 0 \\ 1.519 & 0 & 0 & -5.008e{-}5 \\ 0 & 1.345 & 9.767e{-}6 & 0 \\ 0 & 1.518e{+}5 & 1.345 & 0 \\ -4.607e{+}4 & 0 & 0 & 1.519 \end{array}\right] \quad (248)$$

where the states are

$$\begin{Bmatrix} w_y \\ \theta_y \\ M_y \\ V_y \\ w_z \\ \theta_z \\ M_z \\ V_z \end{Bmatrix}_{end} = \mathbf{U}_{\text{sys}}(s) \begin{Bmatrix} w_y \\ \theta_y \\ M_y \\ V_y \\ w_z \\ \theta_z \\ M_z \\ V_z \end{Bmatrix}_{base} \tag{249}$$

For a clamped-free beam, the boundary conditions are

$$\mathbf{z}_{\text{end}} = \begin{Bmatrix} w_y \\ \theta_y \\ 0 \\ 0 \\ w_z \\ \theta_z \\ 0 \\ 0 \end{Bmatrix} \quad \text{and} \quad \mathbf{z}_{\text{base}} = \begin{Bmatrix} 0 \\ 0 \\ M_y \\ V_y \\ 0 \\ 0 \\ M_z \\ V_z \end{Bmatrix} \tag{250}$$

so that the corresponding **subU** will come from rows 3, 4, 7, and 8 and columns 3, 4, 7, and 8 of $\mathbf{U}_{\text{sys}}(s_1)$:

$$\mathbf{subU}(s_1) = \begin{bmatrix} 1.5189 & 0 & 0 & 5.129 \\ 0 & 1.3451 & -0.29841 & 0 \\ 0 & -4.9683 & 1.3451 & 0 \\ 0.4498 & 0 & 0 & 1.5189 \end{bmatrix} \tag{251}$$

8.3.2.2 Null Space Determination using RREF

Row reducing $\mathbf{subU}(s_1)$ gives

$$RREF(\mathbf{subU}(s_1)) = \begin{bmatrix} 0.29614 & 0 & 0 & 1 \\ 0 & 1 & -0.22185 & 0 \\ 0 & 0 & 0.048878 & 0 \\ 0 & 0 & 0 & O(\varepsilon) \end{bmatrix} \tag{252}$$

where $O(\varepsilon)$ refers to a term whose magnitude is less than 1e–14 and may be considered 0. $\mathbf{subU}(s_1)$ is rank deficient by 1 (i.e. its rank is 3). The basis vector can be found by making an arbitrary choice for V_z (this is equivalent to a mode shape needing some sort of scaling convention) and then back-substituting up the triangular form. The result is

$$\begin{Bmatrix} M_y \\ V_y \\ M_z \\ V_z \end{Bmatrix} = \begin{Bmatrix} -3.3768 \\ 0 \\ 0 \\ 1 \end{Bmatrix} \tag{253}$$

This vector could be normalized to give

$$\begin{Bmatrix} M_y \\ V_y \\ M_z \\ V_z \end{Bmatrix} = \begin{Bmatrix} 0.95884 \\ 0 \\ 0 \\ -0.28395 \end{Bmatrix} \tag{254}$$

Repeating this analysis for the repeated root case ($EI_2/EI_1 = 1$) gives

$$RREF(\mathbf{subU}(s_1)) = \begin{bmatrix} 0.29614 & 0 & 0 & 1 \\ 0 & 1 & -0.29614 & 0 \\ 0 & 0 & O(\varepsilon) & 0 \\ 0 & 0 & 0 & O(\varepsilon) \end{bmatrix} \tag{255}$$

and $\mathbf{subU}(s_1)$ is rank deficient by 2. Care will need to be taken to ensure that the basis vectors are orthogonal to one another. Similar to a generalized eigenvalue problem, once two vectors are found that span this two-dimensional null space, any linear combination of

Table 10: The norm of each row of the $RREF(\mathbf{subU})$.

EI_2/EI_1	$\|row_1\|$	$\|row_2\|$	$\|row_3\|$	$\|row_4\|$
1.5	1.0429	1.0243	4.8878e–2	2.2204e–16
1.1	1.0429	1.0378	1.1818e–2	2.2204e–16
1.01	1.0429	1.0424	1.2384e–3	2.2204e–16
1.001	1.0429	1.0429	1.2444e–4	2.2204e–16
1.0	1.0429	1.0429	5.5511e–17	2.2204e–16

those two vectors is also in the null space. Out of the infinite combinations of null-space basis vectors, an orthogonal pair should be found.

An ortho-normal basis for this **subU** is

$$\begin{Bmatrix} M_y \\ V_y \\ M_z \\ V_z \end{Bmatrix} = \begin{Bmatrix} 0.95884 \\ 0 \\ 0 \\ -0.28395 \end{Bmatrix} \qquad \text{and} \qquad \begin{Bmatrix} M_y \\ V_y \\ M_z \\ V_z \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0.28395 \\ 0.95884 \\ 0 \end{Bmatrix} \tag{256}$$

Table 10 shows the norm of each row of $RREF(\mathbf{subU})$ for all the values of EI_2/EI_1 considered in this analysis. Note that as the system approaches having a repeated root, the second smallest norm approaches 0. When this second smallest norm reaches 0, the rank deficiency of the matrix is two and the null space is two dimensional.

8.3.2.3 Null Space Determination using Eigenvalues/Eigenvectors

The eigenvalues for $\mathbf{subU}(s_1)$ for the case $EI_2/EI_1 = 1.5$ are

$$\lambda = \begin{bmatrix} -1.8182e{-}16 & 0.12746 & 2.5627 & 3.0378 \end{bmatrix} \tag{257}$$

and the corresponding eigenvectors are the columns of

$$\phi = \begin{bmatrix} 0.95884 & O(\varepsilon) & O(\varepsilon) & 0.95884 \\ 0 & 0.23803 & 0.23803 & 0 \\ 0 & 0.97126 & -0.97126 & 0 \\ -0.28395 & O(\varepsilon) & O(\varepsilon) & 0.28395 \end{bmatrix} \tag{258}$$

Note that for the case $EI_2/EI_1 = 1.001$

$$\lambda = \begin{bmatrix} -1.8182e{-}16 & 0.00031912 & 3.0364 & 3.0378 \end{bmatrix} \tag{259}$$

$$\phi = \begin{bmatrix} 0.95884 & 1.5054e{-}13 & 2.4624e{-}12 & 0.95884 \\ 0 & 0.28383 & 0.28383 & 0 \\ 0 & 0.95888 & -0.95888 & 0 \\ -0.28395 & -4.443e{-}14 & 7.287e{-}13 & 0.28395 \end{bmatrix} \tag{260}$$

the eigenvector corresponding to $\lambda = 0.00031912$ is nearly orthogonal to that corresponding to $\lambda = -1.8182e - -16$, but for $EI_2/EI_1 = 1$ the generalized eigenvalue/eigenvector algorithm has lost the orthogonality and care must be taken to restore it:

$$\lambda = \begin{bmatrix} -1.8182e{-}16 & -8.8037e{-}17 & 3.0378 & 3.0378 \end{bmatrix} \tag{261}$$

$$\phi = \begin{bmatrix} 0.95884 & 0.44938 & 0.95884 & 0.95884 \\ 0 & 0.25083 & 0 & 0 \\ 0 & 0.84701 & 0 & 0 \\ -0.28395 & -0.13308 & 0.28395 & 0.28395 \end{bmatrix} \tag{262}$$

For this particular case, this can be done by setting $v_{2ortho} = v_2 - 0.44938/0.95884\ v_1$ and normalizing to give

$$v_{2ortho} = \begin{Bmatrix} 0 \\ 0.28395 \\ 0.95884 \\ O(\varepsilon) \end{Bmatrix} \tag{263}$$

where v_1 and v_2 refer to the first and second columns of ϕ. Note that this approach has led to the same results as the RREF analysis for the repeated root case:

$$\begin{Bmatrix} M_y \\ V_y \\ M_z \\ V_z \end{Bmatrix} = \begin{Bmatrix} 0.95884 \\ 0 \\ 0 \\ -0.28395 \end{Bmatrix} \quad \text{and} \quad \begin{Bmatrix} M_y \\ V_y \\ M_z \\ V_z \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0.28395 \\ 0.95884 \\ O(\varepsilon) \end{Bmatrix} \tag{264}$$

Table 11 shows the eigenvalues of **subU** for all the values of EI_2/EI_1 considered in this analysis. Note that as the system approaches having a repeated root, the second smallest eigenvalue approaches 0. When this second smallest eigenvalue reaches 0, the rank deficiency of the matrix is two and generalized eigenvectors need to be found.

Table 11: Eigenvalues of **subU** for each case of EI_2/EI_1 considered.

EI_2/EI_1	λ_1	λ_2	λ_3	λ_4
1.5	3.0378	2.5627	1.2746e–1	-1.8182e–16
1.1	3.0378	2.9125	3.0360e–2	-3.3795e–16
1.01	3.0378	3.0243	3.1764e–3	-1.8182e–16
1.001	3.0378	3.0364	3.1912e–4	-1.8182e–16
1.0	3.0378	3.0378	-8.8037e–17	-1.8182e–16

8.3.2.4 Null Space Determination using Singular Value Decomposition

Singular value decomposition or SVD may be the least familiar of these approaches for some engineers, but it is the only approach that automatically handles the orthogonality of the basis vectors. For the case $EI_2/EI_1 = 1.5$, the singular values are

$$\lambda^* = \begin{bmatrix} 5.5788 & 5.328 & 0.061306 & 3.1063e{-}17 \end{bmatrix} \tag{265}$$

and the vector corresponding to $\lambda^* = 3.1063e - -17$ is

$$\psi = \begin{Bmatrix} 0.95884 \\ 0 \\ 0 \\ -0.28395 \end{Bmatrix} \tag{266}$$

For the case $EI_2/EI_1 = 1.0$, the singular values are

$$\lambda^* = \begin{bmatrix} 5.5788 & 5.5788 & 3.1063e{-}17 & 3.1063e{-}17 \end{bmatrix} \tag{267}$$

and the basis vectors are

$$\begin{Bmatrix} M_y \\ V_y \\ M_z \\ V_z \end{Bmatrix} = \begin{Bmatrix} 0.95884 \\ 0 \\ 0 \\ -0.28395 \end{Bmatrix} \quad \text{and} \quad \begin{Bmatrix} M_y \\ V_y \\ M_z \\ V_z \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0.28395 \\ 0.95884 \\ 0 \end{Bmatrix} \tag{268}$$

Table 12 shows the singular values of **subU** for all the values of EI_2/EI_1 considered in this analysis. Note that as the system approaches having a repeated root, the second smallest singular value approaches 0. When this second smallest eigenvalue reaches 0, the rank

Table 12: Singular values of **subU** for each case of EI_2/EI_1 considered.

EI_2/EI_1	λ_1^*	λ_2^*	λ_3^*	λ_4^*
1.5	5.5788	5.3280	6.1306e–2	3.1063e–17
1.1	5.5788	5.5096	1.6049e–2	1.4325e–16
1.01	5.5788	5.5712	1.7242e–3	3.1063e–17
1.001	5.5788	5.5781	1.7371e–4	3.1063e–17
1.0	5.5788	5.5788	3.1063e–17	3.1063e–17

deficiency of the matrix is two and the basis vectors for the null space will be the vectors corresponding to the two smallest singular values.

Note that all three methods lead to the same results.

8.3.2.5 Final Steps

Once the non-zero elements of $\mathbf{z}_{\text{base}}$ have been determined, TMM analysis would find the state vector at the free end ($\mathbf{z}_{\text{end}}$) by substituting in the 0 elements from the boundary conditions into $\mathbf{z}_{\text{base}}$ according to equation (250) and multiplying by $\mathbf{U}_{\text{sys}}$ to find $\mathbf{z}_{\text{end}}$:

$$\begin{Bmatrix} w_y \\ \theta_y \\ M_y \\ V_y \\ w_z \\ \theta_z \\ M_z \\ V_z \end{Bmatrix}_{\text{end}} = \begin{Bmatrix} 0 \\ 5.145e{-}6 \\ O(\varepsilon) \\ 0 \\ -1.7374e{-}5 \\ 0 \\ 0 \\ O(\varepsilon) \end{Bmatrix} = \mathbf{U}_{\text{sys}}(s_1) \begin{Bmatrix} 0 \\ 0 \\ 0.95884 \\ 0 \\ 0 \\ 0 \\ 0 \\ -0.28395 \end{Bmatrix} \tag{269}$$

Once $\mathbf{z}_{\text{base}}$ and $\mathbf{z}_{\text{tip}}$ are known, the state vectors at intermediate locations can be calculated in order to visualize the mode shape.

8.3.2.6 Mode Shapes/FEA Comparison

As a final verification that repeated roots are being handled correctly, the first two natural frequencies and mode shapes will be compared with FEA for EI_2/EI_1=1.001 and 1.0. Table 13 compares the eigenvalues from TMM and FEA for the nearly repeated ($EI_2/EI_1 = 1.001$) and repeated ($EI_2/EI_1 = 1.0$) root cases.

Table 13: Comparison of system eigenvalues from TMM and FEA for the nearly repeated and repeated root cases.

EI_2/EI_1	s_1 TMM	s_2 TMM	s_1 FEA	s_2 FEA
1.001	39.597	39.617	39.587	39.607
1	39.597	39.597	39.587	39.587

Figures 148–151 show close agreement between TMM and FEA for the nearly repeated case. Figures 152–155 show that the FEA software used (Dymore) does not return orthogonal modes for the generalized eigenvalue problem.

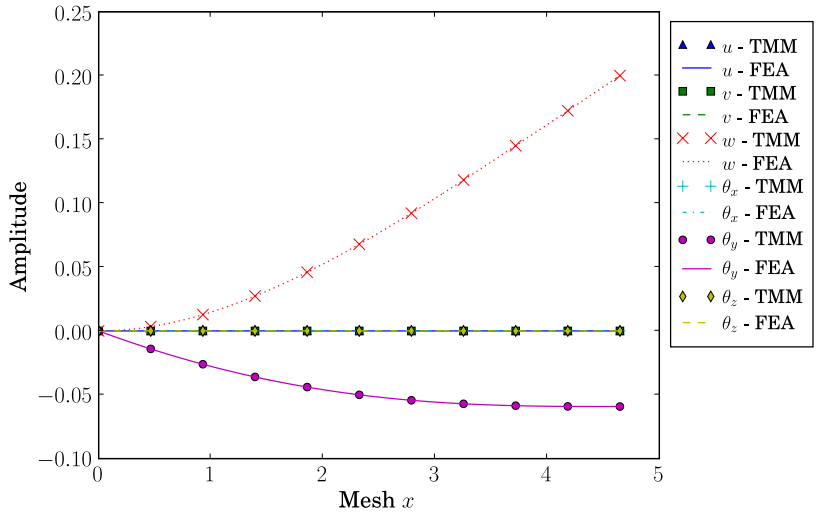

Figure 148: Comparison of mode shapes from TMM and FEA for mode 1 for the case $EI_2/EI_1 = 1.001$.

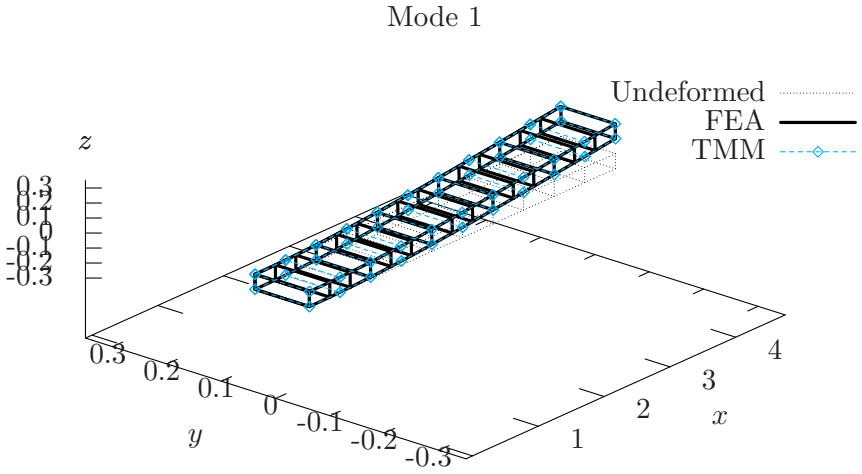

Figure 149: 3D visualization of mode shapes from TMM and FEA for mode 1 for the case $EI_2/EI_1 = 1.001$.

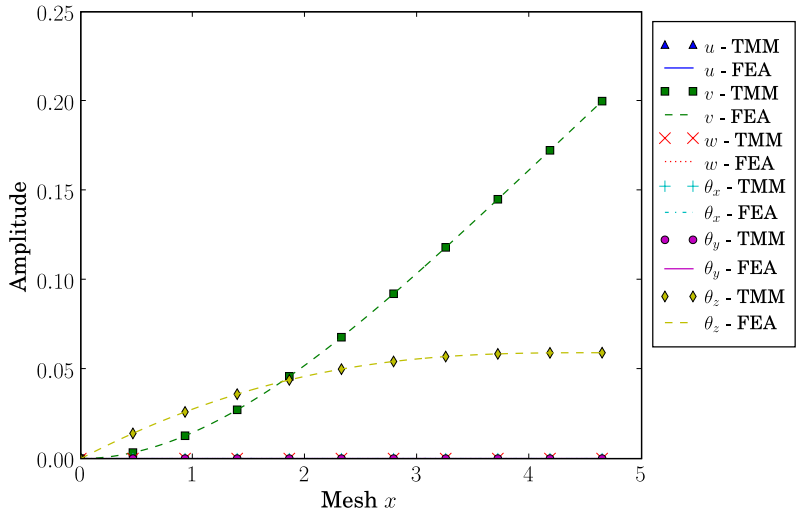

Figure 150: Comparison of mode shapes from TMM and FEA for mode 2 for the case $EI_2/EI_1 = 1.001$.

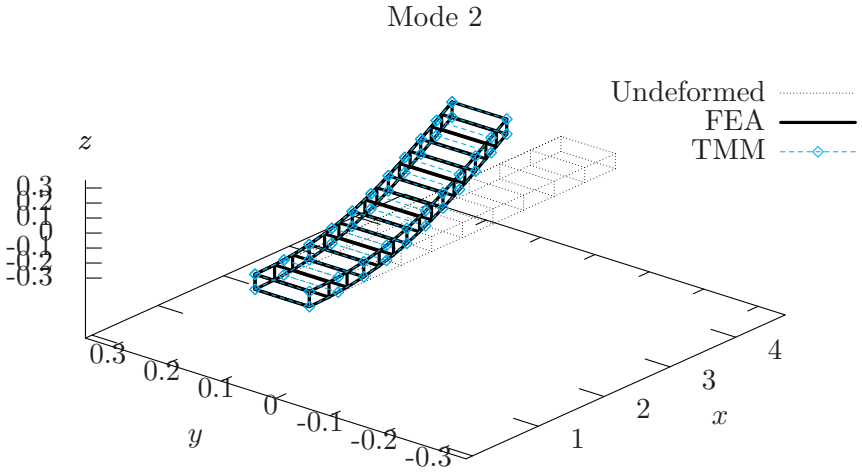

Figure 151: 3D visualization of mode shapes from TMM and FEA for mode 2 for the case $EI_2/EI_1 = 1.001$.

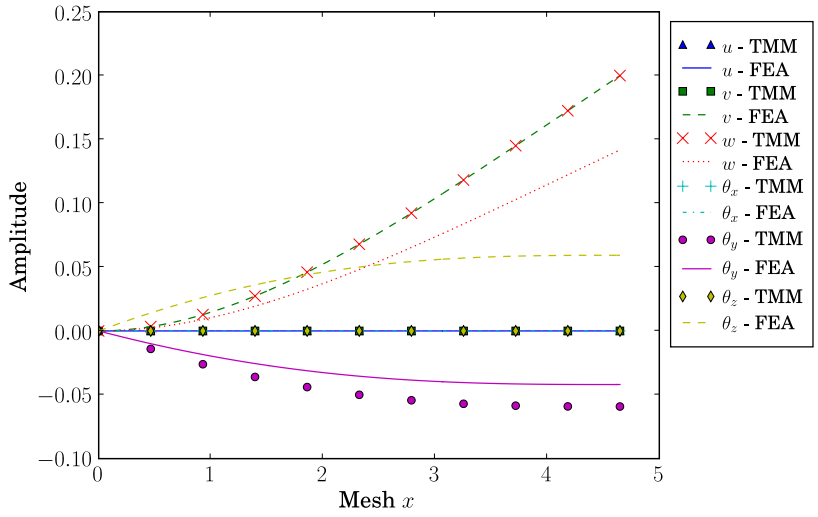

Figure 152: Comparison of mode shapes from TMM and FEA for mode 1 for the case $EI_2/EI_1 = 1.0$.

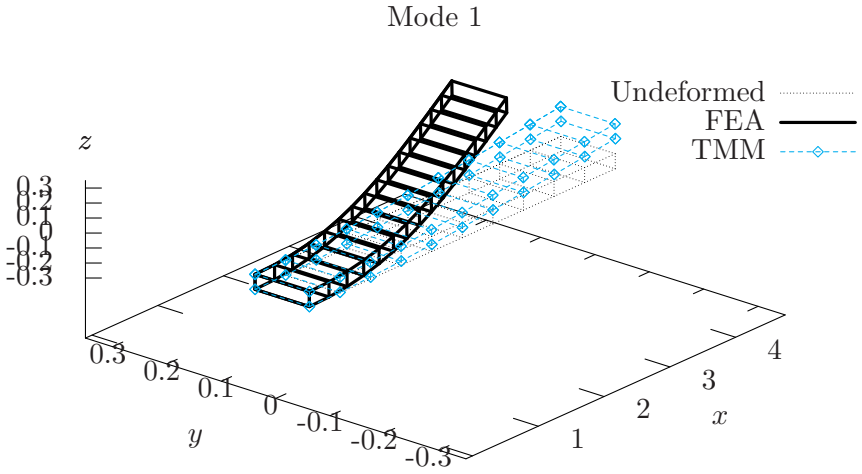

Figure 153: 3D visualization of mode shapes from TMM and FEA for mode 1 for the case $EI_2/EI_1 = 1.0$.

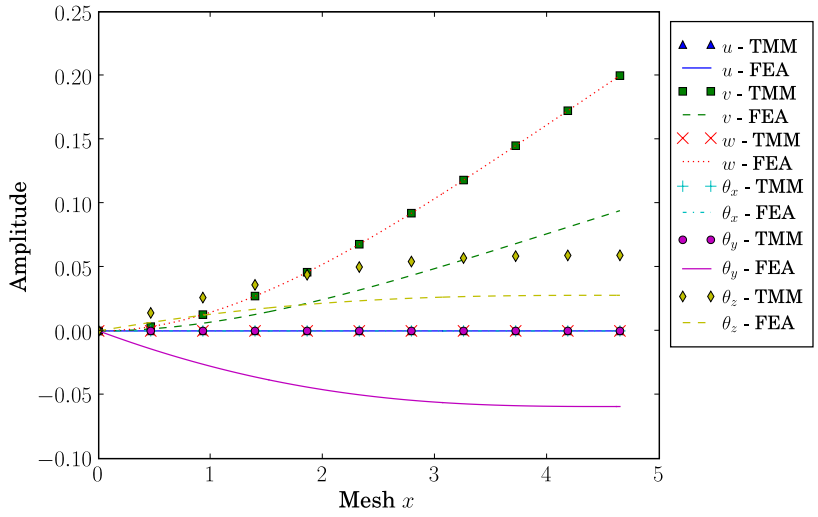

Figure 154: Comparison of mode shapes from TMM and FEA for mode 2 for the case $EI_2/EI_1 = 1.0$.

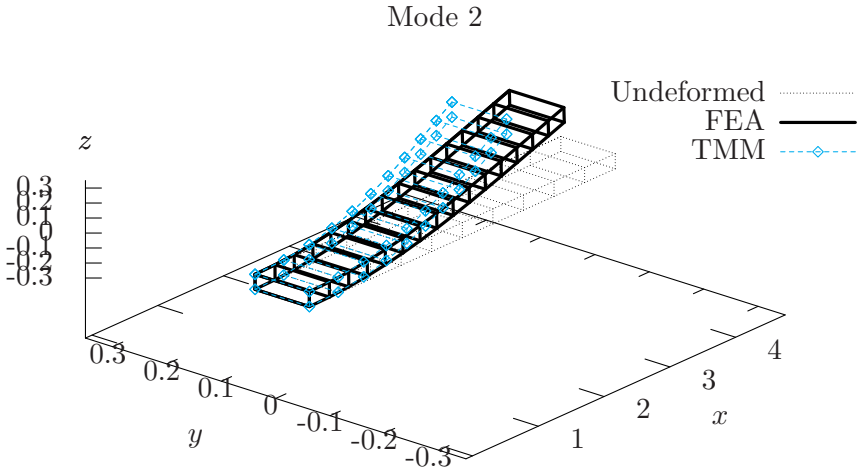

Figure 155: 3D visualization of mode shapes from TMM and FEA for mode 2 for the case $EI_2/EI_1 = 1.0$.

CHAPTER IX

CONCLUSIONS AND RECOMMENDATIONS

9.1 Conclusions

An improved technique for modeling and control design of flexible robots has been developed. This technique is based on the transfer matrix method and is focused on the needs of a controls engineer.

The transfer matrix method has been expanded in several ways to allow it to model flexible robots with complicated actuators and vibration suppression control schemes. A transfer matrix model of a hydraulic actuator has been developed that captures the essential dynamics. This model has been experimentally verified to capture the interaction between the actuator and structure near system resonances.

An approach to modeling non-collocated feedback using the TMM has also been developed. While non-collocated feedback is dangerous from the standpoint of system stability, it is practically unavoidable in some cases. For the experimental testbed used in this work (SAMII), there are two feedback loops where non-collocated control is necessary. The first is the motion control loop where flexibility within the actuator makes it practically impossible to have strictly collocated control. The second is the vibration suppression loop where, in order to have collocated control, accelerometers would have to be installed at every joint. Non-collocated feedback is difficult for the TMM to model because each transfer matrix multiplies the state vector immediately preceding it. There is no means for injecting states from a different spatial location. This limitation of the TMM has been overcome. The approach for modeling non-collocated feedback has been experimentally validated by generating Bode plots of the closed-loop system. Quantitative agreement between model and experiment has been achieved for Bode plots of the actuator and flexible base with θ feedback and with θ plus $\ddot{x}$ feedback. These Bode plots include the first two modes of the

system. Both of these modes lie within the bandwidth of the hydraulic actuator.

Transfer matrices have been derived for the elements necessary to model arbitrary three-dimensional poses of flexible robots, including flexible beams, rigid links, and joints. These matrices have been numerically verified by comparing the natural frequencies and mode shapes for an L-shaped structure with results from finite element analysis.

A key step in moving from a modeling tool to a control design tool was implementing the TMM symbolically. This allows closed-form expressions for the closed-loop system response to be found without modal discretization. This analysis outputs transcendental transfer functions if the system includes continuous elements. Numerical search algorithms can then be used to find the control gains that optimize the closed-loop response. This optimization may be done in terms of the pole locations or the Bode response of the system. The ability to design the controllers without modal discretization is useful whenever feedback modifies the mode shapes of the system. For such a system, all control design approaches that are based on discretized models will need to iterate through the control design and discretization process.

Once closed-form expressions for the closed-loop system response were found, two approaches to control design were pursued. The first was Bode-based optimization. For this approach a cost function was written in terms of quantities that can be measured on Bode plots such as crossover frequency, phase margin, and peak amplitudes. This approach was used to simultaneously optimize the motion control and vibration suppression loops for SAMII based on cost functions associated with the actuator and flexible base Bode plots.

The second control design approach was based on placing or optimizing the closed-loop pole locations. This involved tracking locations of the dominant closed-loop poles as control gains were varied. These poles were the roots of transcendental equations. A pole-tracking algorithm was developed to ensure that all the dominant poles could be found at each step in the optimization. The algorithm included a means to tell which pole was which so that each pole could be plugged into an appropriate part of the cost function. This approach was also used to simultaneously optimize the motion control and vibration suppression loops for SAMII and the results were compared with the Bode-based optimization. The

two approaches led to very similar designs, validating one another. The results were not identical because each approach used a cost function written in terms of variables that are natural for its representation of the system. The Bode-based optimization used crossover frequency and phase margin in its cost function rather than numerical values for the pole locations.

Because the TMM is predominantly focused on the frequency domain, time delays could easily be incorporated into this analysis.

A software package has been developed to perform all of the TMM analysis discussed in this thesis. It is a module in the Python programming language. It makes these analysis tools available to controls engineers who are not experts in the TMM. The software is object oriented has been designed with user extensibility in mind. Hopefully it will provide a useful framework for others to build on.

9.2 Contributions

This thesis has made contributions in the areas of modeling flexible robots, expanding the modeling capabilities of the transfer matrix method, control design using the TMM, analysis of problems relating to the numeric implementation of the TMM, and creating a software package for TMM analysis.

Specific contributions include the following:

1. Developed a hydraulic actuator model that is simple but captures the essential dynamics of the actuator interacting with a flexible robot
    - compatible with the TMM
    - experimentally verified
    - provides better quantitative agreement between model experiment than previous work on modeling hydraulic actuators interacting with flexible structures
2. Developed the transfer matrices necessary to model arbitrary three-dimensional poses of flexible robots and other actuated structures using the TMM
    - numerically verified these matrices through FEA comparison

3. Developed a technique for modeling non-collocated feedback using the TMM
    - this approach leads to quantitative agreement between model and experiment for the closed-loop response of SAMII including higher modes
4. Developed new approaches to control design for flexible robots including multiple feedback loops
    - simultaneous optimization of multiple Bode plots
    - optimized closed-loop pole-locations for a system with two nested control loops
    - developed a robust pole-tracking algorithm to be used with the optimization technique
    - developed a technique for finding closed-form expressions for the closed-loop system response without modal discretization based on symbolic implementation of the TMM
5. Investigated two areas of concern related to the numeric implementation of the TMM
    - demonstrated ways to recognize floating-point error problems and showed that symbolic TMM analysis avoids many of these problems
    - compared three approaches to dealing with repeated system eigenvalues and finding the associated mode shapes
6. Created an object-oriented software package for TMM analysis
    - provides a user extensible framework
    - created the TMMElement and TMMSystem classes and demonstrated their usefulness as software abstractions of physical things

9.3 Recommendations for Future Work

The primary limitations of this work are inherited from the TMM. The approach is inherently linear and is best suited to serial connections of elements. In has been suggested [22] that the TMM could be combined with the describing function method as a first step

toward modeling nonlinear elements. This combined approach could then be used to model hard nonlinearities such as saturation.

This work could be expanded to include more general interconnections of structures including parallel robots.

While this work provides the tools for modeling robots in arbitrary three-dimensional poses, no attempt is made at modeling transitions from one pose to another. This problem could be approached by developing linearized TMM models at various configurations in the workspace and developing a technique for moving from one model to another. Another approach would be to combine a controller developed using the TMM with an augmenting adaptive controller.

It would also be interesting to see how the transfer matrix method and the modeling techniques presented in this thesis could be applied to vastly different problems in other areas of specialization. Using the TMM to predict acoustic radiation of structures would be an interesting problem.

APPENDIX A

DERIVATION OF A CONTINUOUS BEAM ELEMENT

This appendix demonstrates how to derive a transfer matrix for a beam element without discretization (i.e. a continuous beam element). As discussed in section 4.2.1, the state equations for a beam in bending about the z-axis are

$$\theta_z = \frac{dw_y}{dx} \tag{270}$$

$$M_z = EI_z \frac{d^2 w_y}{dx^2} \tag{271}$$

$$V_y = -\frac{\partial M_z}{\partial x} \tag{272}$$

$$\frac{\partial V_y}{\partial x} = \mu \frac{\partial^2 w_y}{\partial t^2} \tag{273}$$

which could be used to derive the partial differential equation

$$-EI_z \frac{\partial^4 w_y}{\partial x^4} = \mu \frac{\partial^2 w_y}{\partial t^2} \tag{274}$$

After separating time and space variation, the spatial differential equation would be

$$W_y'''' - \left(\frac{\beta}{L}\right)^4 W_y = 0 \tag{275}$$

A general solution to equation (275) is

$$W_y = A_1 \sin\left(\frac{\beta x}{L}\right) + A_2 \cos\left(\frac{\beta x}{L}\right) + A_3 \sinh\left(\frac{\beta x}{L}\right) + A_4 \cosh\left(\frac{\beta x}{L}\right) \tag{276}$$

Substituting for w_y in equations 270-272 gives

$$\theta_z = \beta \left[A_1 \cos\left(\frac{\beta x}{L}\right) - A_2 \sin\left(\frac{\beta x}{L}\right) + A_3 \cosh\left(\frac{\beta x}{L}\right) + A_4 \sinh\left(\frac{\beta x}{L}\right)\right] L^{-1} \tag{277}$$

$$M_z = -EI_z \beta^2 \left[A_1 \sin\left(\frac{\beta x}{L}\right) + A_2 \cos\left(\frac{\beta x}{L}\right) - A_3 \sinh\left(\frac{\beta x}{L}\right) - A_4 \cosh\left(\frac{\beta x}{L}\right)\right] L^{-2} \tag{278}$$

$$V_y = EI_z\beta^3\left[A_1\cos\left(\frac{\beta x}{L}\right) - A_2\sin\left(\frac{\beta x}{L}\right) - A_3\cosh\left(\frac{\beta x}{L}\right) - A_4\sinh\left(\frac{\beta x}{L}\right)\right]L^{-3} \quad (279)$$

Note that all states will be multiplied by the time variable $T_y(t)$. Arranging these equations into matrix form gives

$$\begin{Bmatrix} w_y \\ \theta_z \\ M_z \\ V_y \end{Bmatrix} = \begin{bmatrix} \sin\left(\frac{\beta x}{L}\right) & \cos\left(\frac{\beta x}{L}\right) & \cdots \\ \beta\cos\left(\frac{\beta x}{L}\right)L^{-1} & -\beta\sin\left(\frac{\beta x}{L}\right)L^{-1} & \\ -EI_z\beta^2\sin\left(\frac{\beta x}{L}\right)L^{-2} & -EI_z\beta^2\cos\left(\frac{\beta x}{L}\right)L^{-2} & \\ EI_z\beta^3\cos\left(\frac{\beta x}{L}\right)L^{-3} & -EI_z\beta^3\sin\left(\frac{\beta x}{L}\right)L^{-3} & \end{bmatrix}$$

$$\begin{bmatrix} \cdots & \sinh\left(\frac{\beta x}{L}\right) & \cosh\left(\frac{\beta x}{L}\right) \\ & \beta\cosh\left(\frac{\beta x}{L}\right)L^{-1} & \beta\sinh\left(\frac{\beta x}{L}\right)L^{-1} \\ & EI_z\beta^2\sinh\left(\frac{\beta x}{L}\right)L^{-2} & EI_z\beta^2\cosh\left(\frac{\beta x}{L}\right)L^{-2} \\ & -EI_z\beta^3\cosh\left(\frac{\beta x}{L}\right)L^{-3} & -EI_z\beta^3\sinh\left(\frac{\beta x}{L}\right)L^{-3} \end{bmatrix}\begin{Bmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{Bmatrix} \quad (280)$$

Substituting $x = 0$ and $x = L$ into equation (280) gives

$$\begin{Bmatrix} w_y \\ \theta_z \\ M_z \\ V_y \end{Bmatrix}_0 = \begin{bmatrix} 0 & 1 & 0 & 1 \\ \frac{\beta}{L} & 0 & \frac{\beta}{L} & 0 \\ 0 & -\frac{EI_z\beta^2}{L^2} & 0 & \frac{EI_z\beta^2}{L^2} \\ \frac{EI_z\beta^3}{L^3} & 0 & -\frac{EI_z\beta^3}{L^3} & 0 \end{bmatrix}\begin{Bmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{Bmatrix} \quad (281)$$

$$\begin{Bmatrix} w_y \\ \theta_z \\ M_z \\ V_y \end{Bmatrix}_L = \begin{bmatrix} \sin(\beta) & \cos(\beta) & \sinh(\beta) & \cosh(\beta) \\ \frac{\beta\cos(\beta)}{L} & -\frac{\beta\sin(\beta)}{L} & \frac{\beta\cosh(\beta)}{L} & \frac{\beta\sinh(\beta)}{L} \\ -\frac{EI_z\beta^2\sin(\beta)}{L^2} & -\frac{EI_z\beta^2\cos(\beta)}{L^2} & \frac{EI_z\beta^2\sinh(\beta)}{L^2} & \frac{EI_z\beta^2\cosh(\beta)}{L^2} \\ \frac{EI_z\beta^3\cos(\beta)}{L^3} & -\frac{EI_z\beta^3\sin(\beta)}{L^3} & -\frac{EI_z\beta^3\cosh(\beta)}{L^3} & -\frac{EI_z\beta^3\sinh(\beta)}{L^3} \end{bmatrix}\begin{Bmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{Bmatrix} \quad (282)$$

Solving for A_i from equation (281) gives

$$\begin{Bmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{Bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ \frac{\beta}{L} & 0 & \frac{\beta}{L} & 0 \\ 0 & -\frac{EI_z\beta^2}{L^2} & 0 & \frac{EI_z\beta^2}{L^2} \\ \frac{EI_z\beta^3}{L^3} & 0 & -\frac{EI_z\beta^3}{L^3} & 0 \end{bmatrix}^{-1} \begin{Bmatrix} w_y \\ \theta_z \\ M_z \\ V_y \end{Bmatrix}_0 \tag{283}$$

Substituting this expression into equation (282) gives

$$B_{0Lz} = \begin{bmatrix} 1/2\cos(\beta) + 1/2\cosh(\beta) & 1/2\frac{L(\sin(\beta)+\sinh(\beta))}{\beta} & \cdots \\ 1/2\frac{\beta(-\sin(\beta)+\sinh(\beta))}{L} & 1/2\cos(\beta) + 1/2\cosh(\beta) & \\ 1/2\frac{\beta^2(-\cos(\beta)+\cosh(\beta))}{a} & 1/2\frac{\beta L(-\sin(\beta)+\sinh(\beta))}{a} & \\ -1/2\frac{\beta^3(\sin(\beta)+\sinh(\beta))}{La} & -1/2\frac{\beta^2(-\cos(\beta)+\cosh(\beta))}{a} & \end{bmatrix}$$

$$\begin{bmatrix} 1/2\frac{a(-\cos(\beta)+\cosh(\beta))}{\beta^2} & -1/2\frac{La(-\sin(\beta)+\sinh(\beta))}{\beta^3} \\ 1/2\frac{a(\sin(\beta)+\sinh(\beta))}{\beta L} & -1/2\frac{a(-\cos(\beta)+\cosh(\beta))}{\beta^2} \\ 1/2\cos(\beta) + 1/2\cosh(\beta) & -1/2\frac{L(\sin(\beta)+\sinh(\beta))}{\beta} \\ -1/2\frac{\beta(-\sin(\beta)+\sinh(\beta))}{L} & 1/2\cos(\beta) + 1/2\cosh(\beta) \end{bmatrix} \tag{284}$$

where

$$a = \frac{\beta}{L^2} \tag{285}$$

or

$$B_{0Lz} = \begin{bmatrix} c_1 & 1/2\frac{Lc_4}{\beta} & 1/2\frac{ac_3}{\beta^2} & -1/2\frac{Lac_2}{\beta^3} \\ 1/2\frac{\beta c_2}{L} & c_1 & 1/2\frac{ac_4}{\beta L} & -1/2\frac{ac_3}{\beta^2} \\ 1/2\frac{\beta^2 c_3}{a} & 1/2\frac{\beta L c_2}{a} & c_1 & -1/2\frac{Lc_4}{\beta} \\ -1/2\frac{\beta^3 c_4}{La} & -1/2\frac{\beta^2 c_3}{a} & -1/2\frac{\beta c_2}{L} & c_1 \end{bmatrix} \tag{286}$$

where

$$c = \begin{Bmatrix} 1/2\cos(\beta) + 1/2\cosh(\beta) \\ -\sin(\beta) + \sinh(\beta) \\ -\cos(\beta) + \cosh(\beta) \\ \sin(\beta) + \sinh(\beta) \end{Bmatrix} \tag{287}$$

So that $\mathbf{B}_{0Lz}$ is the transfer matrix from $x = 0$ to $x = L$ for bending of a beam about the z axis, or

$$\begin{Bmatrix} w_y \\ \theta_z \\ M_z \\ V_y \end{Bmatrix}_L = \mathbf{B}_{0Lz} \begin{Bmatrix} w_y \\ \theta_z \\ M_z \\ V_y \end{Bmatrix}_0 \tag{288}$$

APPENDIX B

FEA CODE FOR L-SHAPED STRUCTURE

```
@PROCESS_CONTROL_DEFINITION {
  @UNIT_SYSTEM {SI}
  @DEBUG_PRINT_FLAG {      3}
  @FINITE_ELEMENT_ANALYSIS {YES}
  @POST_PROCESSING_ANALYSIS {YES}
  @MANUAL_PATH {C:\dymore2_0\manual\}
  @FIGURES_PATH {figures/}
}
@MODEL_DEFINITION {
@FIXED_FRAME_DEFINITION {
  @FIXED_FRAME_NAME {frame1} {
    @ORIGIN { 0.00000e+000,  0.00000e+000,  0.00000e+000}
    @ORIENTATION_E2 {-1.00000e+000,  0.00000e+000,  0.00000e+000}
    @ORIENTATION_E3 { 0.00000e+000,  0.00000e+000,  1.00000e+000}
  }
}
@TRIAD_DEFINITION {
  @TRIAD_NAME {TriadRvjA} {
    @ORIENTATION_E2 { 0.00000e+000,  1.00000e+000,  0.00000e+000}
    @ORIENTATION_E3 { 0.00000e+000,  0.00000e+000,  1.00000e+000}
  }
  @TRIAD_NAME {TriadRvjB} {
    @ORIENTATION_E2 { 0.00000e+000,  1.00000e+000,  0.00000e+000}
    @ORIENTATION_E3 { 0.00000e+000,  0.00000e+000,  1.00000e+000}
  }
  @TRIAD_NAME {TriadRvjC} {
    @EULER_ANGLES_323 { 0.00000e+000,  5.00000e+000,  0.00000e+000}
  }
  @TRIAD_NAME {TriadRvjD} {
    @ORIENTATION_E2 { 0.00000e+000,  1.00000e+000,  0.00000e+000}
    @ORIENTATION_E3 { 0.00000e+000,  0.00000e+000,  1.00000e+000}
  }
}
@POINT_DEFINITION {
    @POINT_NAME {PointA} {
```

```
        @COORDINATES { 0.00000e+000, 0.00000e+000, 0.00000e+000}
    }
    @POINT_NAME {PointB} {
        @COORDINATES { 4.64820e+000, 0.00000e+000, 0.00000e+000}
    }
    @POINT_NAME {PointC} {
        @COORDINATES { 4.64820e+000, 5.00000e-001, 0.00000e+000}
    }
}
@CURVE_MESH_PARAMETERS_DEFINITION {
    @CURVE_MESH_PARAMETERS_NAME {RigidLinkMesh} {
    @NUMBER_OF_ELEMENTS { 2}
    @ORDER_OF_ELEMENTS { 3}
    }
}
@CURVE_DEFINITION {
    @CURVE_NAME {RigidLinkCurve} {
        @IS_DEFINED_IN_FRAME {frame1}
        @POINT_DEFINITION {
            @NUMBER_OF_CONTROL_POINTS { 2}
            @DEGREE_OF_CURVE { 1}
            @RATIONAL_CURVE_FLAG {NO}
            @END_POINT_0 {PointB}
            @END_POINT_1 {PointC}
        }
        @CURVE_MESH_PARAMETERS_NAME {RigidLinkMesh}
    }
}
@MASS_PROPERTY_DEFINITION {
    @MASS_PROPERTY_NAME {RigidLinkMass} {
        @TOTAL_MASS { 2.00000e+001}
        @CENTRE_OF_MASS_LOCATION { 0.00000e+000, 4.00000e-001, 0.00000e
            +000}
        @MOMENTS_OF_INERTIA { 3.28333e+000, 0.00000e+000, 0.00000e+000,
            4.33333e-001, 0.00000e+000, 3.68333e+000}
    }
}
@RIGID_BODY_DEFINITION {
    @RIGID_BODY_NAME {RigidLink} {
    @CONNECTED_TO_BODY {Beam1}
    @AT_POINT {PointB}
    @CONNECTED_TO_BODY {FreeC}
    @AT_POINT {PointC}
```

```
    @TRIAD_NAME {TRIAD_INERTIAL}
    @MASS_PROPERTY_NAME {RigidLinkMass}
    @CURVE_NAME {RigidLinkCurve}
    @GRAPHICAL_PARAMETERS_NAME {RigidLinkGraphParams}
    }
}
@GRAPHICAL_PARAMETERS_DEFINITION {
    @GRAPHICAL_PARAMETERS_NAME {RigidLinkGraphParams} {
    @OBJECT_HIGHLIGHTED {YES}
    @REPRESENTATION_TYPE {SYMBOL}
    }
}
@BEAM_PROPERTY_DEFINITION {
  @BEAM_PROPERTY_NAME {PropertyBeam1} {
    @ETA_VALUE { 0.00000e+000}{
      @AXIAL_STIFFNESS { 1.45736e+008}
      @BENDING_STIFFNESSES { 3.39137e+005, 5.08706e+005, 0.00000e+000}
      @TORSIONAL_STIFFNESS { 2.55774e+005}
      @SHEARING_STIFFNESSES { 1.30000e+013, 1.30000e+013, 0.00000e
          +000}
      @MASS_PER_UNIT_SPAN { 5.72815e+000}
      @MOMENTS_OF_INERTIA { 2.66596e-002, 1.33298e-002, 1.33298e-002}
      @CENTRE_OF_MASS_LOCATION { 0.00000e+000, 0.00000e+000}
      @SHEAR_CENTRE_LOCATION { 0.00000e+000, 0.00000e+000}
      @CENTROID_LOCATION { 0.00000e+000, 0.00000e+000}
    }
    @COMMENTS {SAMII cantilever beam properties with EI2 changed to get
        different eigenvalues in the two different directions}
  }
}
@BEAM_DEFINITION {
  @BEAM_NAME {Beam1} {
    @CONNECTED_TO_BODY {ClampA}
    @AT_POINT {PointA}
    @CONNECTED_TO_BODY {RigidLink}
    @AT_POINT {PointB}
    @CURVE_NAME {CurveBeam1}
    @BEAM_PROPERTY_NAME {PropertyBeam1}
    @GRAPHICAL_PARAMETERS_NAME {GrfParamBeam1}
  }
}
@CURVE_DEFINITION {
  @CURVE_NAME {CurveBeam1} {
```

```
    @IS_DEFINED_IN_FRAME {INERTIAL}
    @POINT_DEFINITION {
      @NUMBER_OF_CONTROL_POINTS {      2}
      @DEGREE_OF_CURVE {      1}
      @RATIONAL_CURVE_FLAG {NO}
      @END_POINT_0 {PointA}
      @END_POINT_1 {PointB}
    }
    @TRIAD_DEFINITION {
      @ETA_VALUE { 0.00000e+000}
      @ORIENTATION_E2 { 0.00000e+000,  1.00000e+000,  0.00000e+000}
      @ORIENTATION_E3 { 0.00000e+000,  0.00000e+000,  1.00000e+000}
    }
    @CURVE_MESH_PARAMETERS_NAME {MeshBeam1}
  }
}
@CURVE_MESH_PARAMETERS_DEFINITION {
  @CURVE_MESH_PARAMETERS_NAME {MeshBeam1} {
    @NUMBER_OF_ELEMENTS {     50}
    @ORDER_OF_ELEMENTS {      3}
  }
}
@GRAPHICAL_PARAMETERS_DEFINITION {
  @GRAPHICAL_PARAMETERS_NAME {GrfParamBeam1} {
    @REPRESENTATION_TYPE {SURFACE}
    @COLOR_FOR_CONFIGURATION {     0,    255,    255}
    @VECTOR_FIELD_TYPE {FORCES}
    @COLOR_FOR_VECTOR_FIELD {   255,      0,      0}
  }
}
@BOUNDARY_CONDITION_DEFINITION{
    @BOUNDARY_CONDITION_NAME {ClampA} {
        @APPLIED_TO_BODY {Beam1}
        @AT_POINT {PointA}
        @DISPLACEMENT_BOUNDARY_CONDITIONS {1, 1, 1}
        @ROTATION_BOUNDARY_CONDITIONS {1, 1, 1}
    }
    @BOUNDARY_CONDITION_NAME {FreeC} {
        @APPLIED_TO_BODY {RigidLink}
        @AT_POINT {PointC}
        @DISPLACEMENT_BOUNDARY_CONDITIONS {0, 0, 0}
        @ROTATION_BOUNDARY_CONDITIONS {0, 0, 0}
    }
```

```
}
@TIME_FUNCTION_DEFINITION {
  @TIME_FUNCTION_NAME {ScheduleRotationA} {
    @TIME_FUNCTION_TYPE {USER_DEFINED}
    @TABLE_ENTRIES {
      @TIME { 0.00000e+000}  @FUNCTION_VALUE { 0.00000e+000}
      @TIME { 1.00000e+001}  @FUNCTION_VALUE {-2.00000e+002}
    }
  }
}
@SENSOR_DEFINITION {
  @SENSOR_NAME {SensorBeam1Displacements} {
    @OBJECT_NAME {Beam1}
    @SENSOR_TYPE {DISPLACEMENTS}
    @ETA_VALUE { 1.00000e+000}
    @FRAME_NAME {INERTIAL}
  }
  @SENSOR_NAME {SensorBeam1Positions} {
    @OBJECT_NAME {Beam1}
    @SENSOR_TYPE {POSITIONS}
    @ETA_VALUE { 1.00000e+000}
    @FRAME_NAME {INERTIAL}
  }
  @SENSOR_NAME {SensorBeam1Velocities} {
    @OBJECT_NAME {Beam1}
    @SENSOR_TYPE {VELOCITIES}
    @ETA_VALUE { 1.00000e+000}
  }
  @SENSOR_NAME {SensorBeam1Forces} {
    @OBJECT_NAME {Beam1}
    @SENSOR_TYPE {FORCES}
    @ETA_VALUE { 0.00000e+000}
  }
  @SENSOR_NAME {SensorEnergy} {
    @OBJECT_NAME {ENERGY}
  }
}
@SIGNAL_POST_PROCESSING_DEFINITION {
  @SIGNAL_POST_PROCESSING_NAME {SigPostBeam1Forces} {
    @SIGNAL_POST_PROCESSING_TYPE {MINMAX}
    @SENSOR_LIST {
    SensorBeam1Forces}
    @CHANNEL_NUMBER {      3}
```

```
  }
}
}
@CREATE_FINITE_ELEMENT_MODEL {
@CFM_CONTROL_PARAMETERS {
  @ANALYSIS_TYPE {STATIC}
}
}
@FINITE_ELEMENT_ANALYSIS {
@FEM_CONTROL_PARAMETERS {
  @FEM_CONTROL_PARAMETERS_NAME {AnalysisControlParameters} {
    @MAXIMUM_NUMBER_OF_TIME_STEPS {      1}
    @SIMULATION_TIME_RANGE { 0.00000e+000,  0.00000e+000}
    @TIME_STEP_SIZE_RANGE { 1.00000e-007,  1.00000e-002}
    @REFERENCE_ENERGY_LEVEL { 5.00000e+001}
  }
}
@INITIAL_CONDITION_DEFINITION {
}
@STEP_CONTROL_PARAMETERS {
  @STEP_CONTROL_PARAMETER_NAME {Step1} {
    @ARCHIVAL_FREQUENCY {      4}
    @REUSE_NUMBER {100000}
    @NUMBER_OF_EIGENVALUES {     20}
    @EIGENPROBLEM_PRINT_FLAG {      0}
    @GYROSCOPIC_TERMS {NO}
    @EIGEN_SPECTRUM_SHIFT { 0.00000e+000}
    @SPECTRAL_RADIUS_AT_INFINITY { 0.00000e+000}
    @TIME_STEP_SIZE { 0.00000e+000}
    @MAXIMUM_NUMBER_OF_ITERATIONS {     10}
    @FACTORIZATION_STRATEGY {      5}
    @CONVERGENCE_TOLERANCE { 1.00000e-006}
    @MAXIMUM_NUMBER_OF_REJECT {      5}
    @AVERAGE_STIFFNESS_TERM { 1.00000e+008}
    @AVERAGE_MASS_TERM { 0.00000e+000}
  }
}
}
@POST_PROCESSING_ANALYSIS {
@GRAPHICS_CONTROL_PARAMETERS {
  @GRAPHICS_CONTROL_PARAMETERS_NAME {GraphicsControlParameters} {
    @TIME_STEP_SIZE { 0.00000e+000}
    @EIGENVECTORS_SCALING_FACTOR { 2.00000e-001}
```

```
    @MODAL_ANIMATION_CYCLES {       5}
    @MODAL_ANIMATION_FRAMES_PER_CYCLE {      50}
    @VECTOR_FIELD_TYPE {FORCES}
    @VECTOR_FIELD_SCALING_FACTOR { 1.00000e+000}
    @VECTOR_FIELD_TYPE {VELOCITIES}
    @VECTOR_FIELD_SCALING_FACTOR { 1.00000e+000}
  }
}
@VIEW_PARAMETERS_DEFINITION {
  @VIEW_PARAMETERS_NAME {ViewParameters} {
    @VIEW_REFERENCE_POINT { 5.00000e-001,  5.00000e-001,  5.00000e-001}
    @VIEW_SIZE { 1.20000e+000,  1.20000e+000,  1.20000e+000}
    @PROJECTION_REFERENCE_POINT { 1.66233e-001, -8.33333e-001, -8.33333e
        -001}
    @PROJECTION_EYE_VECTOR { 6.91358e-001, -6.91358e-001,  2.09877e-001}
    @PROJECTION_UP_VECTOR { 3.95062e-001,  6.04938e-001,  6.91358e-001}
    @PROJECTION_VIEWPORT {-2.00747e+000,  2.00747e+000, -1.10370e+000,
        1.10370e+000}
  }
}
}
```

APPENDIX C

PYTHON CODE

C.1 TMM.TMMElement

```
class TMMElement:
    """This is the base class for creating TMMElement models.  It is not
        intended to be used directly.  Users must derive their own
       classes from it, and derived classes must override GetMat and
       GetHT."""
    def __init__(self,elemtype,params,maxsize=12,label='',symname='',
       symlabel='',symsub=False,unknownparams=None,functionparams=None,
       compensators=None):
        """Create a TMMElement instance.  Since the class is not
           intended to be used without deriving from it, this code
           should be called from each derived methods __init__ method.
            It contains code that should be common to all
           __init__methods for all TMMElement classes."""

    def GetMat(self, s, sym=False):
        """This method must be overridden in all derived TMMElement
           classes.  It returns the element transfer matrix given 's'
           as an input.  It should be defined in a way that allows it
           to output either a numeric or symbolic transfer matrix.  For
            symbolic output, 's' should be a symstr."""

    def GetAugMat(self, s, sym=False, maxlen=100):
        """This method returns the augmented element transfer matrix.
           By default, it returns an N+1 by N+1 matrix with with self.
           GetMat substituted into the upper left N by N matrix.  This
           is exactly what is needed for unactuated elements.
           Unactuated elements do not need to override this method.
           Actuated elements must override this method.  Provided that
           GetMat is written in a way that is compliant with either
           numeric or symbolic output, this method will be as well."""

    def GetHT(self):
```

```
        """This method returns the homogeneous transformation matrix for
            the TMMElement.  This matrix is used in three dimensional
           visualization of mode shapes.  This method must be
           overridden in derived classes."""

    def SubstituteUnknowns(self, unknowndict):
        """This method is used to substitute the results from system
           identification into the proper place in the TMM model.  The
           input is a dictionary containing unknown parameters and
           their values from system i.d.  This function looks for the
           unknown parameters of this element in the unknown dictionary
            and substitutes the appropriate values.  If all of the
           unknownparams of the element are found and substituted, then
            self.unknownparams is set to None."""

    def GetSymMat(self):
        """This function is called by self.GetMaximaLines to get the
           symbolic element transfer function.  By default, it simply
           calls self.GetMat(s, sym=True) where s is a symstr.  This
           function could be overridden by a derived element if for
           some reason self.GetMat cannot be written in such a way that
            in can output either a symbolic or numeric transfer matrix.
             As long as self.GetMat can return a symbolic transfer
           matrix for the element, this function does not need to be
           overridden by derived elements."""

    def GetAugSymMat(self):
        """Similar to GetSymMat, this method returns a symbolic transfer
            matrix for the element.  By default, it calls self.
           GetAugSymMat(s, sym=True) with s as a symstr."""

    def GetMaximaLines(self,aug=False):
        """This is the new (as of 04/04/06) method for symbolic analysis
            based on the GetSymMat and GetAugSymMat methods.  This
           function outputs a list that can be appended to a latexlist
           as part of the input to the Python-Maxima-Latex symbolic
           engine."""

    def parsetype(self, elemtype):
        """This method assigns a numeric value for the element type and
           is called by the __init__ method."""

    def GetComplexList(self,label=None):
```

```
        """This function returns a list of all variables associated with
             an element that would need to be declared complex in a
            FORTRAN function that includes the element in a Bode model.
             This function is used in symbolic work.  This is the
            default function defined in the top level TMMElement class.
             It assumes that most elements have only real parameters and
             returns an empty list."""

    def GetFortranDefs(self):
        """This method is used to specify definitions that need to go at
             the beginning of automatically generated FORTRAN functions,
             so that FORTRAN analysis of the element will go smoothly.""
            "


class TMMElementLHT(TMMElement):
    """This class is meant to be a base class for all elements whose
       homogeneous transformation matrix performs only a translation
       along the x-axis by the system parameter 'L' - i.e.\  that
       elements do not rotate the coordinate system used for
       visualization.  This class should be used as the base class for
       beam and rigid link elements.  It is a convenience class - it
       saves the author of new TMMElement classes from having to define
        the GetHT method for the new element."""
    def GetHT(self):
        """Return the homogeneous transformation matrix for the
           TMMElementLHT class.  This matrix will represent a
           translation along the 'x' axis with no rotation."""


class TMMElementIHT(TMMElement):
    """This is another convenience class meant to be a base class for
       torsional springs and other elements whose homogeneous
       transformation matrix is simply and identity matrix."""
    def GetHT(self):
        """Return a 4x4 identify matrix for the homogeneous
           transformation matrix of the TMMElementIHT class."""
```

C.1.1 Module Functions

```
def HT4(axis='',angle=0,x=0.,y=0.,z=0.):
```

```
    """This function returns a 4x4 homogeneous transformation matrix.
       Axis is either 'x', 'y', or 'z' (strings).  If axis is empty (if
        bool(axis) if false), there is no rotation portion to the
       matrix and the upper left 3x3 will be an identity matrix.  angle
        is in degrees.  x, y, and z are the translations associated
       with the homogeneous transformation matrix."""

def DH(d,theta,a,alpha):
    """Return a Denavit-Hartenburg homogeneous transformation matrix
       according to the conventions of Sciavicco and Siciliano, second
       edition, p.  45 This means translating by d about the z_{i-1}
       and rotated by theta about that same axis.  Then translate by a
       along the x' axis and rotate by alpha about the x' axis.  Alpha
       and theta are both in degrees,"""

def rot3by3(axis,angle):
    """Generate a 3x3 rotation matrix rotating angle degrees about axis.
       """

def rotationmat(axis,angle):
    """Create a 12x12 rotation matrix for use with 3D, 12x12 TMM
       analysis."""

def Transform8by8(matin):
    """This function takes in an 8x8 matrix and switches rows and
       columns to unshuffle the mixed states associated with 2 and 3D
       transfer matrices."""
```

C.2 TMM.spring

```
class TorsionalSpringDamper(TMMElementIHT):
    """This class represents a spring element with compliance along 1,
       2, or 3 axes (for maxsize = 4, 8, or 12)."""
    def __init__(self,params,**kwargs):
        """Create a TorsionalSpringDamper instance.  params is a
           dictionary with keys of 'k' and 'c'. 'k' and 'c' are lists
           or arrays if maxsize > 4.  If maxsize==8, k = [k0,k1].  If
           maxsize==12, k=[k0, k1, k2]. 'c' is optional and will be set
            to zeros(shape(k)) if it is omitted.  Otherwise c=c0, [c0],
            [c0,c1], or [c0, c1, c2]."""

    def GetMat(self,s,sym=False):
```

```
        """Return the element transfer matrix for the
           TorsionalSpringDamper element.  If sym=True, 's' must be a
           symbolic string and a matrix of strings will be returned.
           Otherwise, 's' is a numeric value (probably complex) and the
            matrix returned will be complex."""


class Spring2x2(TMMElementIHT):
    """This class represents a spring element in a system where only
       linear displacement and force are used as states.  This would be
        used with mass-spring-damper simplified systems like a system
       of carts used in a lot of introductory vibrations or controls
       classes."""
    def __init__(self,params,symlabel='K',**kwargs):
        """Create an instance of the Spring2x2 class.  params is a
           dictionary for consistency with other TMMElement's, but it
           has only one entry: 'k'."""

    def GetMat(self,s,sym=False):
        """Return the transfer matrix for the Spring2x2 TMMElement,
           given s as an input.  If sym=True, the transfer matrix will
           be a string."""
```

C.3 TMM.beam

```
class BeamElement(TMMElementLHT):
    """This class represents a continuous beam element.  It can be 1, 2,
        or 3 dimensional.  In the 3D case, the x-axis includes axial
       and torsional vibration."""
    def __init__(self,params,**kwargs):
        """Create a continuous beam element.  params should be a
           dictionary with keys 'mu', 'L', and 'EI'.  If the beam is
           more than 1 dimensional (i.e.\  if maxsize!=4), than params
           should specify 'EI1' and 'EI2' as well as 'EA', 'm11', 'GJ'
           if axial and torsional analysis is to be performed.  kwargs
           must include {'usez':True} if a 4x4 beam element should
           model bending about the z-axis.  The y-axis is used if 'usez
           ' is False or not specified."""

    def GetMat(self,s,sym=False):
```

```
        """Return the element transfer matrix for the BeamElement
           element.  If sym=True, 's' must be a symbolic string and a
           matrix of strings will be returned.  Otherwise, 's' is a
           numeric value (probably complex) and the matrix returned
           will be complex."""

    def GetHT(self):
        """Return the homogeneous transformation matrix for the
           TMMElementLHT class.  This matrix will represent a
           translation along the 'x' axis with no rotation."""

    def GetComplexList(self,label=None):
        """This function returns a list of all variables associated with
            a beam bending about one axis that would need to be
           declared complex in a FORTRAN function that includes a beam
           in a Bode model.  This function is used in symbolic work."""

    def GetMaximaSubstitutions(self,name=None,label=None,aug=0):
        """This function is used to substitute sinh and cosh terms into
           the symbolic matrix so that round off problems can be
           avoided by symbolic manipulation."""
```

C.3.1 Module Functions

```
def cos(ent):
    """Polymorphic cos function that adapts to either a numeric or
       symbolic string input."""

def cosh(ent):
    """Polymorphic cosh function that adapts to either a numeric or
       symbolic string input."""

def sin(ent):
    """Polymorphic sin function that adapts to either a numeric or
       symbolic string input."""

def sinh(ent):
    """Polymorphic sinh function that adapts to either a numeric or
       symbolic string input."""

def bendmaty(s,params,EIstr='EI1'):
```

```
    """Return a 4x4 transfer matrix for a beam in bending about the y-
       axis.  This function is used for part of the GetMat function of
       an 8x8 or 12x12 beam.  It will be the entire 4x4 transfer matrix
        if usez=False for the beam element."""

def bendmatz(s,params,EIstr='EI2',symlabel=''):
    """Return a 4x4 transfer matrix for a beam in bending about the z-
       axis.  This function is used for part of the GetMat function of
       an 8x8 or 12x12 beam.  It will be the entire 4x4 transfer matrix
        if usez=True for the beam element."""

def axialtorsionmat(s,params):#L,mu,EA,m11,GJ):
    """Return a 4x4 transfer matrix for axial and torsional vibration of
        a beam about its x-axis.  This function is used for part of the
        GetMat function of a 12x12 beam."""

def torsionmat(s,params):#L,m11,GJ):
    """Return the torsional vibration 2 by 2 transfer matrix used as
       part of axialtorsionmat."""

def axialmat(s,params):#L,mu,EA):
    """Return the axial vibration 2 by 2 transfer matrix used as part of
        axialtorsionmat."""
```

C.4 TMM.rigid

```
class RigidMass(TMMElementLHT):
    """This class models a rigid mass element in a TMM model.  It can be
        1, 2, or 3 dimensional."""
    def __init__(self,params,**kwargs):
        """Create a rigid mass element.  params should be a dictionary
           with keys 'L', 'm', 'r', and 'I'. 'L' is the length of the
           element. 'm' is the mass of the element. 'r' is the distance
            to the center of gravity of the link. 'I' is the second
           moment of inertia and it should be a vector if this is a 2
           or 3D link.  kwargs should contain {'usez':True} if a 4x4
           rigid mass element should model rotation about the z-axis.""
           "

    def GetMat(self, s, sym=False):
```

```
        """Return the element transfer matrix for the RigidMass element.
             If sym=True, 's' must be a symbolic string and a matrix of
            strings will be returned.  Otherwise, 's' is a numeric
           value (probably complex) and the matrix returned will be
           complex."""


class RigidMass2x2(TMMElementLHT):
    """The class models a 2 by 2 mass element where the states are
       displacement and force.  Used to model carts in simple mass-
       spring-damper systems."""
    def __init__(self,params,symlabel='M',**kwargs):
        """Create a 2 by 2 mass element.  params is a dictionary with
           only one key: params={'m':##.##} (the mass of the cart)."""

    def GetMat(self,s,sym=False):
        """Return the 2 by 2 transfer matrix for a cart mass element."""
```

C.4.1 Module Functions

```
def rigidmatx(s,params):
    """Return the 4x4 transfer matrix for motion of a rigid mass about
       its x-axis."""

def rigidmatz(s,params):
    """Return the 4x4 transfer matrix for motion of a rigid mass about
       its z-axis."""

def rigidmaty(s,params):
    """Return the 4x4 transfer matrix for motion of a rigid mass about
       its y-axis."""
```

C.5 TMM.velocitysource

```
class AngularVelocitySource(TMMElementLHT):
    """This class models an angular velocity source.  This class is used
        to model rotary hydraulic actuators."""
    def __init__(self,params={},**kwargs):
```

```
        """Initialize an instance of the AngularVelocitySource class.
           params is a dictionary with keys 'K', 'tau', and 'axis'.
           All of these keys are optional. 'K' is the gain and it
           defaults to 1. 'axis' is the axis about which the actuator
           rotates - it defaults to 1. 'tau' is the pole of the first
           order lag of the actuator (i.e.\  if 'tau' is given and is >
            0, the transfer function of the actuator will be tau/(s*(s+
           tau))."""

    def GetMat(self, s, sym=False):
        """Return the element transfer matrix for the
           AngularVelocitySource element.  If sym=True, 's' must be a
           symbolic string and a matrix of strings will be returned.
           Otherwise, 's' is a numeric value (probably complex) and the
            matrix returned will be complex.  Note that the
           AngularVelocitySource element will return an identity matrix
            for its non-augmented transfer matrix."""

    def GetAugMat(self, s, sym=False):
        """Return the augmented element transfer matrix for the
           AngularVelocitySource element, which includes the velocity
           source portion of 1/s in the augmented column for theta.  If
            sym=True, 's' must be a symbolic string and a matrix of
           strings will be returned.  Otherwise, 's' is a numeric value
            (probably complex) and the matrix returned will be complex.
           """

    def GetHT(self):
        """Return the homogeneous transformation matrix for the
           TMMElementLHT class.  This matrix will represent a
           translation along the 'x' axis with no rotation."""


class AVSwThetaFB(TMMElementLHT):
    """This class models the closed-loop response of an angular velocity
        source with compliance and relative theta feedback."""
    def __init__(self,params={}, **kwargs):
```

```
        """Initialize the AVSwThetaFB instance.  params should have the
           following keys: 'ks' - spring constant (required) 'c' -
           damper coefficient (defaults to 0) 'Ka' - actuator gain (
           defaults to 1) 'Gc' - proportional feedback gain (defaults
           to 1) (Gc can be a transfer function modeled as a ratio of
           polynomials when passed to FORTRAN.) 'axis' - axis about
           which the actuator rotates (defaults to 1) 'tau' - first
           order pole of actuator."""

    def GetMat(self,s):
        """Return the element transfer matrix for the AVSwThetaFB
           element.  If sym=True, 's' must be a symbolic string and a
           matrix of strings will be returned.  Otherwise, 's' is a
           numeric value (probably complex) and the matrix returned
           will be complex.  Note that the AngularVelocitySource
           element will return an identity matrix for its non-augmented
            transfer matrix."""

    def GetAugMat(self, s, sym=False):
        """Return the augmented element transfer matrix for the
           AVSwThetaFB element.  If sym=True, 's' must be a symbolic
           string and a matrix of strings will be returned.  Otherwise,
            's' is a numeric value (probably complex) and the matrix
           returned will be complex."""

    def GetComplexList(self,label=None):
        """This function returns a list of all variables associated with
            this element that would need to be declared complex in a
           FORTRAN function that includes the element in a Bode model.
            This function is used in symbolic work."""
```

C.6 TMM.forcing

```
class ForcingElement(TMMElementLHT):
    """This class represents arbitrary forcing at any location in a TMM
       model, using a vector as a column of the augmented transfer
       matrix."""
    def __init__(self,params,maxsize=12,label=''):
        """Initialize the ForcingElement.  params is a dictionary with
           only one key: 'fv', whose value is a column vector of the
           elements of the augmented column of the transfer matrix.  It
            should have only N element (the bottom 1 is assumed)."""

    def GetMat(self,s):
```

```
        """Return the element transfer matrix for the ForcingElement
           element.  If sym=True, 's' must be a symbolic string and a
           matrix of strings will be returned.  Otherwise, 's' is a
           numeric value (probably complex) and the matrix returned
           will be complex.  Note that the ForcingElement element will
           return an identity matrix for its non-augmented transfer
           matrix."""

    def GetAugMat(self,s):
        """Return the augmented element transfer matrix for the
           ForcingElement element.  If sym=True, 's' must be a symbolic
            string and a matrix of strings will be returned.  Otherwise
           , 's' is a numeric value (probably complex) and the matrix
           returned will be complex."""
```

C.7 TMM.feedback

```
class SAMIIAccelFB(TMMElementLHT):
    """The class represents the accelerometer feedback element for SAMII
       .  It is a very specific class that will not be much use in
       other models."""
    def __init__(self,link0,joint1,link1,avs,Ga=1,axis=1,maxsize=4,label
       ='',symlabel='accelfb',symname='Uaccelfb',compensators='Ga',**
       extraargs):
        """Initializes the accelerometer feedback element.  link0,
           joint1, link1, and avs are all TMMElement's that must be
           passed in.  Their parameters are used in creating the
           transfer matrix for acceleration feedback."""

    def GetSymAugMat(self,s):
        """Return the augmented element transfer matrix for the
           SAMIIAccelFB element.  If sym=True, 's' must be a symbolic
           string and a matrix of strings will be returned.  Otherwise,
            's' is a numeric value (probably complex) and the matrix
           returned will be complex."""

    def GetComplexList(self):
        """This function returns a list of all variables associated with
            this element that would need to be declared complex in a
           FORTRAN function that includes the element in a Bode model.
            This function is used in symbolic work."""

    def GetAugMat(self,s,myparams=None):
```

```
        """Return the augmented element transfer matrix for the
           SAMIIAccelFB element.  If sym=True, 's' must be a symbolic
           string and a matrix of strings will be returned.  Otherwise,
            's' is a numeric value (probably complex) and the matrix
           returned will be complex."""
```

C.8 TMM.TMMSystem

```
class TMMSubSystem(list):
    def invert(self):


class TMMSystem:
    def __init__(self,elemlist,bcend=[2,3,6,7],bcbase=[0,1,4,5],bodeouts
        =[]):

    def PreparePythonFiles(self, filenames,chardetnames=[],ratmx=False,
        curvefit=True):
        """Similar to PrepareFortranFiles, but this function generates
           python files instead of FORTRAN files.  The input is a list
           of auto-generated FORTRAN files (raw FORTRAN files output by
            Maxima).  If these were processed with ratmx=True, then a
           list of chardetnames may also be passed.  These functions
           would contain expressions for the characteristic determinant
            of the transfer function.  The output is a list of python
           filenames and a vector of the initial guesses, which come
           from the parameter values declared in the model for the
           unknownparams."""

    def CleanPythonFiles(self, filenames,frlistin=[]):

    def CallF2py(self, finalfnames):
        """This function takes as input a list of FORTRAN files called
           finalfnames which are ready to compile FORTRAN files
           presumably generated by PrepareFortranFiles.  the '_out.f'
           is stripped off of each filename and an 'f' is appended to
           create the module name.  f2py is then called with os.system
           ('f2py -c -m '+modulename+' '+filename) for each file in
           finalfnames.  The list of modulenames is returned."""

    def GetAllFortranDefs(self):

    def PrepareFortranFiles(self, filenames,headername=None,**kwargs):
```

```
    """This function takes a list of FORTRAN filenames which are
       files that were auto-generated by Maxima (raw FORTRAN files)
       .  It gets parameters and other definitions from the
       TMMSystem model and calls rwkfortran.MakeFortranFunction to
       create FORTRAN files that are ready to be compiled with f2py
        - including adding headers.  The output is a list of the
       ready-to-compile filenames."""

def GetAllComplexVariables(self):
    """This function calls GetComplexList for each element in self.
       elemlist in order to return a list of all the complex
       variables that will need to be declared in a FORTRAN
       function generated by Maxima."""

def GenMaxima(self,prefix='',skipMaxima=True,showbode=False, newsym=
    True, **kwargs):
    """This file generates a LaTeX file that describes the TMMSystem
       .  This file can be used as an input to the Python->Maxima->
        LaTeX symbolic engine.  Optionally, if skipMaxima=False,
       Maxima will be called, Python or FORTRAN files will be
       generated and an output LaTeX file will be created that
       contains the symbolic results from Maxima.  The return
       values are outname, bodenames, chardetnames."""

def RunMaxima(self, texname,frlist=[]):
    """Takes as an input the name of a LaTeX file and runs the
       symbolic Python->Maxima->LaTeX engine on that file.  The
       output of that file is saved to raw FORTRAN files which will
        need to have headers added to them and some other parsing
       done before they can be compiled (FORTRAN_ or imported (
       Python).  PreparePythonFiles and PrepareFortranFiles take
       care of this additional parsing.  The resulting LaTeX
       outputs from Maxima are used to replace the maxima
       environments in the input LaTeX file and an output LaTeX
       file is generated.  The return value is the name of that
       output LaTeX file."""

def GetAllFunctionParams(self):

def IntegratedCurveFit(self,expmodname='',prefix='',fitfortran=True,
    skipMaxima=False,**kwargs):
```

```
        """This function runs an automated system identification process
            for TMMSystem objects in three steps: 1.  Create Python and
           /or FORTRAN functions for the Bode responses of the system.
            These functions will have the names prefix+'bode'+n, where
           the integer n is from iterating over the defined bodeouts of
            the TMMSystem.  2.  Prepare the experimental data for easy
           use with the cost function of the curve-fitting routine.
           This is done by taking the name of a module containing
           experimental rwkbode instances, searching for matches to the
            input/output pairs defined in the bodeouts of the model,
           and creating a list of exp.  bodes from these matches and
           saving the list to a new module.  Saved and loaded modules
           come from the scipy.io.save method that shelves a dictionary
           .  3.  Auto-generate a script that will actually run the the
            minimization that does the curve fitting.  prefix will be
           used in a number of other places for filenames and such.
           kwargs will be passed to SymBodeMaximaAll and consist of the
            following: texname='SymBodeDev.tex' frlist=[] basebodename
           ='bode' basecdname='chardet' debug=False grind=True optimize
           =True skipMaxima=False"""

    def CreateFortranandPythonModules(self,prefix,runcompile=True,
        newsym=True):
        """This function generates the Maxima code, runs Maxima,
           prepares the Fortran files, and optionally calls F2Py.  It
           essentially does the initial steps of IntegratedCurveFit,
           but does not step up the experimental data nor create the
           cost functions."""

    def FindExpData(self,modulename,freqnames=['freq','expfreq'],
        bodenames=['bodes','bode','expbodes','expbode'],datadir=None,
        prefix='exp'):
```

```
    """Import the module named modulename and find the Bodes in it
       that match the bodeouts of the TMMSystem model.  modulename
       is assumed to be a module saved using the scipy.io.save
       method that contains a frequency vector modulename.freqname.
         A list of Bodes is expected to be found at modulename.
       bodename where freqname and bodename are one of the values
       in freqnames and bodenames respectively.  They will be tried
        in order and a break will be executed once one is found
       using hasattr.  The method returns a dictionary whose keys
       are prefix+'freq' (the frequency vector) and prefix+'bodes'
       (a list of Bodes in the same order as self.bodeouts).  If
       datadir is specified, it will be appended to sys.path if it
       is not already in it.  Alternately, modulename can be a
       dictionary instead of a module name.  If so, it should have
       keys named freqname and bodename1 or bodename2."""

def PrepareExpData(self,modulename,prefix='',**otherargs):

def AutoGenerateFitFunction(self,savedname,bodenames,functionname=
    None,initialguesses=[],prefix='',fortran=False):

def SymCharDet(self, sysmatname='Usys',outpath='chardet.f',sub=True,
    showsub=True,radcan=False):
    """This function calls SymSubmat to find the submatrix and then
       calculates its determinant to find the characteristic
       determinant of the system.  Note that you must call
       SymULatex first."""

def SymCharDet(self, sysmatname='Usys',outpath='chardet.f',sub=True,
    showsub=True,radcan=False):
    """This function calls SymSubmat to find the submatrix and then
       calculates its determinant to find the characteristic
       determinant of the system.  Note that you must call
       SymULatex first."""

def SymDefineAllUs(self,aug=0,**kwargs):

def SymBodeAll(self,**kwargs):
```

```
        """This method is the new, improved approach to symbolic Bode
           analysis as of April 4, 2006.  It relies on the new symstr
           based GetAugSymMat methods which are typically inherited
           from the base TMMElement class.  This method generates a *.
           tex file that is an input file for the Python+Maxima+LaTeX
           symbolic engine.  This input file defines a transfer matrix
           for each element in the system as well as a system transfer
           matrix and the bode outputs.  If (not skipMaxima), Maxima is
            called and files will be created for each bode output and
           characteristic determinant.  If optimize=True, these output
           files will be FORTRAN files, otherwise they will be *.txt
           files.  The output the filename of a latexlist containing
           the Maxima lines, a list of the raw FORTRAN files for the
           Bodes, and a list of raw FORTRAN files for the
           characteristic determinants."""

    def SymUsys(self,sysmatname='Usys',frlist=[],stopind=None,intro=None
        ,showU=False,**unusedargs):
        """This is the new method for outputting the Maxima lines that
           calculate the system transfer matrix, assuming the element
           matrices have already been defined in Maxima/LaTeX.  This
           method replaces SymULatex.  It returns a list of lines to be
            appended to a Maxima/Latex list."""

    def SymBodeMaximaAll(self,**kwargs):
        """This method generates a *.tex file that is an input file for
           the Python+Maxima+LaTeX symbolic engine.  This input file
           defines a transfer matrix for each element in the system as
           well as a system transfer matrix and the bode outputs.  If (
           not skipMaxima), Maxima is called and files will be created
           for each bode output and characteristic determinant.  If
           optimize=True, these output files will be FORTRAN files,
           otherwise they will be *.txt files."""

    def LoadExpBodeData(self,modulename,datadir):

    def ExtractInitialGuesses(self):

    def GetUnkownParams(self):

    def GetCompensators(self):

    def SubstituteUnknowns(self, unknowndict):
```

```
def BuildUnknownDict(self, ucv):

def NewPythonFilewithSubs(self, filename, unknowndict, sublist,
    prefix='my',headername='myheader.py'):

def GetParamsandDefs(self,curvefit=1,otherparams={},prefersubs=0,aug
    =0):

def SymBodes(self,basebodename='bodeoutputs',basecdname='chardet',
    myext='.txt',debug=False,optimize=True,ratmx=False,showbode=
    False,sub=True,**unusedargs):
    """This method converts rawbodeouts (which are just the states
        at specified model locations) into actual model simulations
        of measured signals.  This is done by calculating the
        difference between to rawbodeouts for differential sensors,
        by multiplying by gains when needed, and by multiplying by s
         or s**s for velocity or acceleration sensors."""

def SymBodes(self,basebodename='bodeoutputs',basecdname='chardet',
    myext='.txt',debug=False,optimize=True,ratmx=False,showbode=
    False,sub=True,**unusedargs):
    """This method converts rawbodeouts (which are just the states
        at specified model locations) into actual model simulations
        of measured signals.  This is done by calculating the
        difference between to rawbodeouts for differential sensors,
        by multiplying by gains when needed, and by multiplying by s
         or s**s for velocity or acceleration sensors."""

def SymBodesAllinOne(self,texname='SymBodeDev.tex',frlist=[],
    basebodename='bode',basecdname='chardet',functionparams={},
    otherparams={},debug=False,grind=True,optimize=True,skipMaxima=
    False,ratmx=False):

def FindSymRawBode(self,pind,dof):
    """This function returns the symbolic name for a raw bode output
         matching a specific location and dof.  It is assumed that
        you have already run SymRawBodes to calculate the rawbode
        outputs and attach their symbolic names."""

def SymRawBodes(self, frlist=[], funcname='SymULatex'):
```

```
    """This function calculates the raw bode outputs for a TMMSystem
        .  It assumes that the base vector bv is already defined, so
         you must have already called SymBaseVect.  This function
        returns a LaTeX list that calculates the matrices U_i that
        transfer from the base vector to the raw bode locations as
        well as the raw bode outputs themselves where rb_i=U_i*bv By
         specifying funcname='SymUsys', this function is compatible
        with the new symbolic functions."""

def SymBaseVect(self,N=None,showall=False,ratmx=False,**unusedargs):
    """This function finds the base vector used to calculate the
        system bode response.  It assumes that SymSubmat has already
         been called and will append lines using submat and subcol,
        assuming they have already been defined by SymSubmat called
        with findsubcol=True.  This function returns lines to be
        append to a LaTeX list.  These lines create a symbolic
        solution for the base vector."""

def SymSubmat(self, sysmatname='Usys',findsubcol=False,N=None,
    showsub=False,sub=False):
    """Note that you must call SymUsys or SymULatex first.  This
        function will return lines that are to be appended to a
        latexlist that already includes the SymUsys or SymULatex
        lines.  This function uses the system boundary conditions to
         find a sub-matrix whose determinant is the characteristic
        equation of the system."""

def MaximaSubstitute(self,namebefore,nameafter=''):

def _parsebcstr(self, bcstr):

def CreateSysHT(self, beammesh=10):

def CreateMesh(self, beammesh=10):

def FindModeShape(self, eigval,beammesh=10,logtex=0,fmt='0.5g',scale
    =True,modenum=0):

def FindBaseVect(self, eigval,usesvd=True):

def BodeResponse(self,fvect):
```

```
        """Calculate the Bode response at system locations specified by
           self.bodeouts (specified during __init__ of the TMMSystem).
            fvect is a vector of frequencies at which the system
           response will be calculated.  Each entry in fvect is assumed
            to be a real number in Hz."""

    def FindRawBodeout(self,pind,dof):

    def FindAugBaseVect(self,s):

    def FindEig(self, guess,mytol=1e-14,maxfun=1e4,maxiter=1e3,
       returncomplex=False,useabs=True):

    def EigError(self, value,useabs=True):

    def FindU(self,value):

    def FindAugU(self,value):

    def FindSubmat(self,value,eps=1e-14):

    def FindSubmatwAugcol(self,value):

    def FindAugSubmat(self,value):

    def BaseVectBode(self, wvect):

    def GetValuesorDefaults(self,dictin,keylist):
        """This is a convenience function for passing values to lots of
           different subfunctions.  dictin is used to update the
           defaults dict specified in this function.  keylist is then
           used to retrieve the desired values from the update dict.
           The return value is the list of values."""
```

C.8.1 Module Functions

```
def Getkwargs(dictin,keylist):

def null(A, eps=1e-10):

def minindex(listin):

def norm2(x):
    """compute |x|^2 = x*conjugate(x)"""
```

```
def matmax(matin):

def matmaxi(matin):

def matmaxabs(matin):

def CleanInitLine(linein):

def ReplaceSubSystemCalls(linein, subsysnames):

def GetClassNamefromInspectList(listin):

def ReplaceClassName(listin, oldname, newname):

def ProcessOneSubstitutionLine(linein, symsublist, unknowndict):

def GetBodeouts(listin):

def GetUnknownBodeouts(listin, unknowndict):

def ProcessBodeoutLine(linein, gaindict):
```

```
import TMMElement
import TMMSystem

import beam, feedback, spring, forcing, velocitysource, rigid

def DeepReload():
    reload(TMMElement)
    reload(TMMSystem)
    reload(beam)
    reload(feedback)
    reload(spring)
    reload(forcing)
    reload(velocitysource)
    reload(rigid)

%%

%% This is file `listings.sty',

%% generated with the docstrip utility.

%%
```

REFERENCES

[1] Alberts, T. E., *Augmenting the Control of a Flexible Manipulator with Passive Mechanical Damping*. PhD thesis, Georgia Institute of Technology, School of Mechanical Engineering, Sept. 1986.

[2] Banerjee, J. R., "Dynamic stiffness formulation for structural elements: a general approach," *Computers & Structures(UK)*, vol. 63, no. 1, pp. 101–103, 1997.

[3] BANERJEE, J. R. and FISHER, S. A., "Coupled bending-torsional dynamic stiffness matrix for axially loaded beam elements," *International Journal for Numerical Methods in Engineering*, vol. 33, pp. 739–751, 1992.

[4] Bascetta, L. and Rocco, P., "Modelling flexible manipulators with motors at the joints," *Mathematical and Computer Modelling of Dynamical Systems*, vol. 8, pp. 157–183, Jun 2002.

[5] Bauchau, O. A., "DYMORE User's Manual," tech. rep., Georgia Institute of Technology, 2006. http://www.ae.gatech.edu/people/obauchau/DYMORE.html, [cited June 21, 2006].

[6] Bendat, J. S. and Piersol, A. G., *Engineering Applications of Correlation and Spectral Analysis*. New York, NY: John Wiley & Sons, 1980.

[7] Book, W. J., "Recursive lagrangian dynamics of flexible manipulator arms.," *International Journal of Robotics Research*, vol. 3, pp. 87–101, Fall 1984.

[8] Book, W. J., Maizza-Neto, O., and Whitney, D. E., "Feedback control of two beam, two joint systems with distributed flexibility.," *Journal of Dynamic Systems, Measurement and Control, Transactions of the ASME*, vol. 97 Ser G, pp. 424–431, Dec 1975.

[9] Book, W. J. and Majette, M. W., "Controller design for flexible, distributed parameter mechanical arms via combined state space and frequency domain techniques," *Journal of Dynamic Systems, Measurement and Control, Transactions ASME*, vol. 105, no. 4, pp. 245 – 254, 1983.

[10] Book, W. J., *Modeling, Design and Control of Flexible Manipulator Arms*. PhD thesis, Massachusetts Institute of Technology, Apr. 1974.

[11] Book, W. J., Majette, M. W., and Ma, K., "Distributed systems analysis package (dsap) and its application to modeling flexible manipulators," tech. rep., NASA, July 1979. Final Report, Subcontract No. 551 to Charles Stark Draper Laboratory, NASA Contract NAS9-13809.

[12] Book, W. J., Majette, M. W., and Ma, K., "Frequency domain analysis of the space shuttle manipulator arm and its payloads," tech. rep., NASA, February 1981. Final Report, Subcontract No. 586 to Charles Stark Draper Laboratory, NASA Contract NAS9-13809.

[13] Brüls, O., *Integrated Simulation and Reduced-Order Modeling of Controlled Flexible Multibody Systems*. PhD thesis, University of Liège, Belgium, 2005.

[14] Brüls, O., Duysinx, P., and Golinval, J.-C., "A unified finite element framework for the dynamic analysis of controlled flexible mechanisms," in *Proc. of the ECCOMAS Conf. on Advances in Computational Multibody Dynamics*, (Madrid, Spain), June 2005.

[15] Brüls, O., Duysinx, P., and Golinval, J.-C., "The global modal parameterization for nonlinear model-order reduction in flexible multibody dynamics," *Int. J. Num. Meth. Eng.*, vol. in press, pp. –, 2006.

[16] Calise, A. J., Prasad, J. V. R., and Siciliano, B., "Design of optimal output feedback compensators in two-time scale systems," *IEEE Transactions on Automatic Control*, vol. 35, pp. 488–492, Apr 1990.

[17] Calise, A. J., Yang, B.-J., and Craig, J. I., "An augmenting adaptive approach to control of flexible systems," in *Proceedings of AIAA Guidance, Navigation, and Control Conference*, vol. AIAA-2002-4942, 2002.

[18] Cannon, D. W., "Command generation and inertial damping control of flexible macro-micro manipulators," Master's thesis, Georgia Institute of Technology, May 1996.

[19] Chalhoub, N. G. and Chen, L., "A structural flexibility transformation matrix for modelling open-kinematic chains with revolute and prismatic joints," *Journal of Sound and Vibration*, vol. 218, pp. 45–63, Nov 1998.

[20] Deluca, A. and Siciliano, B., "Closed-form dynamic-model of planar multilink lightweight robots," *IEEE Transactions on Systems Man and Cybernetics*, vol. 21, pp. 826–839, Jul-Aug 1991.

[21] Dorsey, J., *Continuous and Discrete Control Systems*. Boston: McGraw-Hill, 2002.

[22] Ferri, A., 2005. private communication.

[23] George, L. E., *Active Vibration Control of a Flexible Base Manipulator*. PhD thesis, Georgia Institute of Technology, School of Mechanical Engineering, Aug. 2002.

[24] HALLAUER, W. L. and L., L. R. Y., "Beam bending-torsion dynamic stiffness method for calculation of exact vibration modes," *Journal of Sound and Vibration*, vol. 85, pp. 105–113, 1982.

[25] Hastings, G. G. and Book, W. J., "Experiments in optimal control of a flexible arm," in *Proceedings of the 1985 American Control Conference*, pp. 728–9 Vol.2, IEEE, 1985.

[26] Hisseine, D. and Lohmann, B., "Robust control for a flexible-link manipulator using sliding mode techniques and nonlinear h_∞ control design methods," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 4, pp. 3865–3870, 2001.

[27] Huggins, J. D., "Experimental verification of a model of a two-link flexible, lightweight manipulator," Master's thesis, Georgia Institute of Technology, School of Mechanical Engineering, June 1988.

[28] Inman, D. J., *Engineering Vibration*. Englewood Cliffs, NJ: Prentice Hall, 1994.

[29] Kang, B. S. and Mills, J. K., "Dynamic modeling of structurally-flexible planar parallel manipulator," *Robotica*, vol. 20, pp. 329–339, May-Jun 2002.

[30] Kao, C. K. and Sinha, A., "Sliding mode control of vibration in flexible structures using estimated states," *Proceedings of the American Control Conference*, vol. 3, pp. 2467–2474, 1991.

[31] Krauss, R., Brüls, O., and Book, W., "Two competing linear models for flexible robots: Comparison, experimental validation, and refinement," in *Proceedings of the 2005 American Control Conference*, (Portland, OR), June 2005.

[32] Kwon, D.-S. and Book, W. J., "A time-domain inverse dynamic tracking control of a single-link flexible manipulator," *Transactions of the ASME. Journal of Dynamic Systems, Measurement and Control*, vol. 116, no. 2, pp. 193 – 200, 1994.

[33] Lagarias, J., Reeds, J., Wright, M., and Wright, P., "Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions," *SIAM Journal of Optimization*, vol. 9, no. 1, pp. 112–147, 1998.

[34] Lahdhiri, T. and Elmaraghy, H. A., "Design of an optimal feedback linearizing-based controller for an experimental flexible-joint robot manipulator," *Optimal Control Applications and Methods*, vol. 20, no. 4, pp. 165–182, 1999.

[35] Lee, J. W., *Dynamic Analysis and Control of Lightweight Manipulators with Flexible Parallel Link Mechanisms*. PhD thesis, Georgia Institute of Technology, June 1990.

[36] Lew, J. Y. and Book, W. J., "Hybrid control of flexible manipulators with multiple contact," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2, pp. 242–247, 1993.

[37] Loper, J. C., "Vibration cancellation and disturbance rejection in serially linked micro/macro menipulators," Master's thesis, Georgia Institute of Technology, Mar. 1998.

[38] Luo, Z. H., "Direct strain feedback control of flexible robot arms: new theoretical and experimental results," *IEEE Transactions on Automatic Control*, vol. 38, no. 11, pp. 1610–1622, 1993.

[39] Lutz, M. and Ascher, D., *Learning Python*. Sebastopol,CA: O'Reilly, second ed., 2003.

[40] Magee, D. P., *Optimal Arbitrary Time-Delay Filtering to Minimize Vibration in Elastic Manipulator Systems*. PhD thesis, Georgia Institute of Technology, Aug. 1996.

[41] Majette, M. W., "Modal state variable control of a linear distributed mechanical system modeled with the transfer matrix method," Master's thesis, Georgia Institute of Technology, School of Mechanical Engineering, June 1985.

[42] McKinnon, K., "Convergence of the Nelder-Mead Simplex Method to a Nonstationary Point," *SIAM Journal on Optimization*, vol. 9, no. 1, pp. 148–158, 1998.

[43] Megahed, S. M. and Hamza, K. T., "Modeling and simulation of planar flexible link manipulators with rigid tip connections to revolute joints," *Robotica*, vol. V 22, pp. P 285–300, May/June 2004.

[44] Morris, A. S. and Madani, A., "Quadratic optimal control of a two-flexible-link robot manipulator," *Robotica*, vol. 16, pp. 97–108, 1998.

[45] Nagaraj, B. P., Nataraju, B. S., and Ghosal, A., "Dynamics of a two-link flexible system undergoing locking: Mathematical modelling and comparison with experiments," *Journal of Sound and Vibration*, vol. 207, pp. 567–589, Nov 1997.

[46] Nelder, J. A. and Mead, R., "A simplex method for function minimization," *Computer Journal*, vol. 7, pp. 308–313, 1965.

[47] Obergfell, K., *End-Point Position Sensing and Control of Flexible Multi-Link Manipulators*. PhD thesis, Georgia Institute of Technology, School of Mechanical Engineering, Aug. 1998.

[48] Obergfell, K., *End-Point Position Sensing and Control of Flexible Multi-Link Manipulators*. PhD thesis, Georgia Institute of Technology, Aug. 1998.

[49] Pestel, E. C. and Leckie, F. A., *Matrix Methods in Elastomechanics*. New York: McGraw Hill, 1963.

[50] Piedboeuf, J. C. and Moore, B., "On the foreshortening effects of a rotating flexible beam using different modeling methods," *Mechanics of Structures and Machines*, vol. 30, no. 1, pp. 83–102, 2002.

[51] Reisenauer, B. T., Balas, M. J., and Ramey, M., "Reduced-order model based control of large flexible manipulators: Theory and experiments," *Proceedings of the American Control Conference*, pp. 1760–1765, 1990.

[52] Rocco, P. and Book, W. J., "Modelling for two-time scale force/position control of flexible robots," in *Proceedings. 1996 IEEE International Conference on Robotics and Automation*, vol. 3, pp. 1941–1946, 1996.

[53] Sano, H., "Residual mode filters and h_∞ control for a class of infinite-dimensional discrete-time systems," *International Journal of Systems Science*, vol. 34, pp. 205–214, Feb 20 2003.

[54] Siciliano, B. and Book, W. J., "Singular perturbation approach to control of lightweight flexible manipulators.," *International Journal of Robotics Research*, vol. 7, pp. 79–90, Aug 1988.

[55] Siciliano, B. and Book, W. J., "Singular perturbation approach to control of lightweight flexible manipulators.," *International Journal of Robotics Research*, vol. 7, pp. 79–90, Aug 1988.

[56] Subudhi, B. and Morris, A. S., "Dynamic modelling, simulation and control of a manipulator with flexible links and joints," *Robotics and Autonomous Systems*, vol. 41, pp. 257–270, Dec 2002.

[57] Tokhi, M. O., Mohamed, Z., and Shaheed, M. H., "Dynamic characterisation of a flexible manipulator system," *Robotica*, vol. 19, pp. 571–580, Sep-Oct 2001.

[58] Torby, B. J. and Kimura, I., "Dynamic modeling of a flexible manipulator with prismatic links," *Journal of Dynamic Systems Measurement and Control-Transactions of the ASME*, vol. 121, pp. 691–696, Dec 1999.

[59] Xu, J. X. and Cao, W. J., "Active vibration control of a single-link flexible manipulator by pole-placement approach," *Proceedings of the 38th IEEE Conference on Decision and Control*, vol. 4, pp. 3888–3893, 1999.

[60] Yang, B., "Transfer functions of constrained/combined one-dimensional continuous dynamic systems," *Journal of Sound and Vibration*, vol. 156, no. 3, 1992.

[61] Yang, B., *Stress, Strain, and Structural Dynamics*. Elsevier Academic Press, 2005.

[62] Yang, B.-J., *Adaptive Output Feedback Control of Flexible Systems*. PhD thesis, Georgia Institute of Technology, School of Mechanical Engineering, Apr. 2004.

[63] Yang, B.-J., Calise, A. J., and Craig, J. I., "Adaptive output feedback control with input saturation," in *Proceeding of the American Control Conference*, vol. 2, pp. 1572–1577, June 2003.

[64] Yu, J.-F., Lien, H.-C., and Wang, B. P., "Exact dynamic analysis of space structures using timoshenko beam theory," *AIAA Journal*, vol. 42, pp. 833–839, Apr 2004.

[65] Yuan, B. S., J., W., and Huggins, J. D., "Dynamics of flexible manipulator arms: Alternative derivation, verification, and characteristics for control," *Journal of Dynamic Systems, Measurement and Control, Transactions of the ASME*, vol. 115, pp. 394–404, Sep 1993.

[66] Yuan, B. S., J., W., and Siciliano, B., "Direct adaptive control of a one-link flexible arm with tracking," *Journal of Robotic Systems*, vol. 6, no. 6, pp. 663–80, 1989.

[67] Zhou, T., Zu, J. W., and Goldenberg, A. A., "Modeling of flexible robot-payload systems through component synthesis," *Journal of Dynamic Systems Measurement and Control-Transactions of the ASME*, vol. 122, pp. 381–386, Jun 2000.

VITA

Ryan W. Krauss was born in Grand Rapids, MI in March of 1974. He earned his bachelor's degree in mechanical engineering from Michigan Technological University in 1996. He earned a master's degree in engineering mechanics from Virginia Polytechnic Institute and State University in 1998.

In the fall semester of 1998, Ryan taught as an adjunct faculty in the mechanical engineering deparment of Grand Valley State University. From 1999–2002, he worked as a safety test engineer for Johnson Controls, Inc. Ryan began his Ph.D. studies at Georgia Tech in the Fall of 2002.

Ryan married Missy Dickson in May of 2000. It was the smartest thing he ever did. Ryan and Missy are moving to Edwardsville, Illinois where Ryan will be an assistant professor in the mechanical engeering deparment at Southern Illinois University Edwardsville.